2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

PIMan: A Comprehensive Approach for Establishing Plausible Influence among Software Repositories

Md Omar Faruk Rokon
UC Riverside
mroko001@ucr.edu

Risul Islam
UC Riverside
risla002@ucr.edu

Md Rayhanul Masud
UC Riverside
mmasu012@ucr.edu

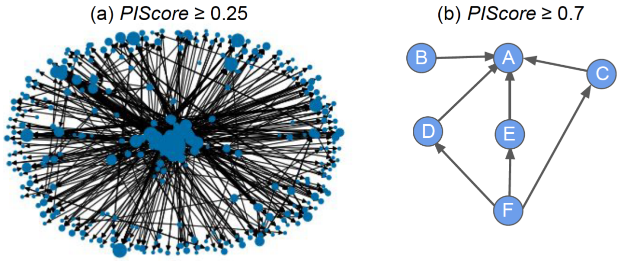
Michalis Faloutsos
UC Riverside
michalis@cs.ucr.edu

***Abstract*—How can we quantify the influence among repositories in online archives like GitHub? Determining repository influence is an essential building block for understanding the dynamics of GitHub-like software archives. The key challenge is to define the appropriate representation model of influence that captures the nuances of the concept and considers its diverse manifestations. We propose PIMan, a systematic approach to quantify the influence among the repositories in a software archive by focusing on the *social* level interactions. As our key novelty, we introduce the concept of Plausible Influence which considers three types of information: (a) repository level interactions, (b) author level interactions, and (c) temporal considerations. We evaluate and apply our method using 2089 malware repositories from GitHub spanning approximately 12 years. First, we show how our approach provides a powerful and flexible way to generate a plausible influence graph whose density is determined by the Plausible Influence Threshold (*PIT*), which is modifiable to meet the needs of a study. Second, we find that there is a significant collaboration and influence among the repositories in our dataset. We identify 28 connected components in the plausible influence graph ($PIT = 0.25$) with 7% of the components containing at least 15 repositories. Furthermore, we find 19 repositories that influenced at least 10 other repositories directly and spawned at least two "families" of repositories. In addition, the results show that our influence metrics capture the manifold aspects of the interactions that are not captured by the typical repository popularity metrics (e.g. number of stars). Overall, our work is a fundamental building block for identifying the influence and lineage of the repositories in online software platforms.**

***Index Terms*—Empirical, Plausible Influence, GitHub Repository, Social Interaction, Collaboration.**

I. Introduction

Determining the influence among software repositories is an essential building block for studying the dynamics of software evolution and collaboration in online Open Source Software (OSS) platforms. These OSS platforms contain a massive number of repositories and facilitate the engagement of millions of users [1]. GitHub is arguably the largest such platform with more than 32 million repositories and 34 million users exhibiting significant collaborative interactions [2]. As these are open source coding platforms, there are no restrictions for users to create new repositories. Naturally, it attracts users with varied expertise levels, and they develop their own software, and also copy and duplicate other repositories. As a result, there is a significant collaboration and code reuse [3], [4] in these platforms. Researchers are interested in studying the dynamics of the ecosystem at the repository level, which could reveal insights into software evolution. Interestingly, there is also a significant malware development activity within public repositories, which could be valuable for security analysts [5], [6]. Therefore, we decide to focus on establishing influence among malware repositories in our work.


(a) *PIScore* ≥ 0.25

(b) *PIScore* ≥ 0.7

Fig. 1: Our approach captures plausible influence between repositories in a tuneable and visually powerful way. We show: (a) the dense PIGraph with lower influence threshold, $PIScore \geq PIT = 0.25$ with 426 nodes, and 1191 edges, and (b) the sparse PIGraph with higher influence threshold, $PIScore \geq PIT = 0.7$ with 6 nodes, and 7 edges.

Given two repositories, how can we quantify the level of influence between them by analyzing their platform-level interactions? This is the problem that we address here. We can identify the following types of interactions: (a) an author can "appreciate" the repository of another author by starring it, watching it, and commenting on it, (b) an author can *follow* another author, and (c) an author can fork popular repositories in the archive. The key challenge is developing a comprehensive approach for defining influence, and even further, estimating the possibility that such an influence has occurred. Combining these different types of information is a non-trivial task. Note that here we define influence which is a much broader concept than, say, code-level similarity which obviously overlaps with influence but is ultimately different. In fact, our intention is to explore the interplay between influence and code-level similarity. Establishing similarity at the code level is an open research question in its own right and here we only employ it as an indirect validation of our influence metric in this study. Once we can quantify the influence between two repositories, we can understand the influence interactions for a group of repositories.

The challenge: the elusive nature of influence. We want to stress that the goal is to identify the **likelihood** of influence

IEEE/ACM ASONAM 2022, November 10-13, 2022
http://dx.doi.org/10.1145/XXXXXXX.XXXXXXX 978-1-6654-5661-6/22/$31.00 © 2022 IEEE

between repositories as it is hard to prove influence with certainty. First, the concept of influence is inherently challenging and goes beyond the code-level similarity. For example, author A can be inspired by repository R, even copy it initially, but then improve it substantially. The final repository can have minimal code level similarity with the repository R. Second, it is nearly impossible to establish influence even if we define it in a very strict sense: author A copies (parts of) repository R of author B. For example, we can think of a scenario where authors A and B emulate or copy a third repository (or some other source). So even if two copied repositories are identical, that does not prove that one has influenced the other in a strict sense. However, we argue that the likelihood of influence will be very helpful in studying software evolution and collaboration patterns.

There are relatively few efforts that focus on establishing influence among repositories. Most efforts typically focus on one or a few high-level metrics such as forking relationships and number of stars, or the focus on influence and popularity of authors. First, there are efforts that study the popularity and importance of repositories [7], [8] using a limited number of readily available metrics such as star and fork relationships. However, these works do not focus on the likelihood of influence between a pair of repositories. Second, there are studies [5], [6], [9] that focus on the author-author level interactions and popularity and not on the repository-repository level. Finally, there are efforts that study code-level similarity, which we view as complementary to our work. We discuss related works in more detail in Section VI.

As our key contribution, we propose PIMan[1], a comprehensive multi-dimensional approach to identify pair-wise **plausible influence** at the repository level. First, our method quantifies the pair-wise influence of repositories considering most social-interaction dimensions comprehensively: (a) repository level interactions, (b) author level interactions, and (c) temporal considerations. Second, our method generates a Plausible Influence Graph (**PIGraph**) for a group of repositories, where an edge between two repositories exists if the **Plausible Influence Score** (***PIScore***) of these repositories is greater than the **Plausible Influence Threshold** (***PIT***). Our approach is a comprehensive and flexible way to combine the multifaceted information using either our default or customized parameter values, tuned to match the niche needs of study.

We deploy our approach on a dataset of 2089 malware repositories from GitHub [5] and study the influence relationship of its repositories. These repositories have been created during a span of approximately 12 years which can capture the long-term effects and phenomena. Although our approach can be applied to any type of repository, focusing on malware repositories: (a) hones in on a rather well-defined community, and (b) could manifest practical value in combating cybercrime. Our key results can be summarized in the following points.

a. PIMan models influence flexibly with a directed graph. Our approach captures repository-level influence relationships with a flexible and informative **plausible influence graph (PIGraph)**. In Figure 1, we illustrate the descriptive power of our approach by looking at different levels of influence using the *PIT* threshold. For $PIT = 0.25$, we get a dense graph of 426 nodes and 1191 edges, whereas for $PIT = 0.7$, the graph contains 6 nodes and 7 edges. By tuning this threshold, we can hone in on different intensity levels of influence. In addition, we show that our *PIScore* is significantly different from other straightforward metrics of popularity.

b. Plausible influence as a proxy for code-level similarity. We show that our definition of influence correlates with code-level similarity as shown in Figure 3 (Spearman coefficient, $\rho = 0.79$, and $p - value = 1.26e - 19$). We consider the following as an indirect validation of our approach: (a) our quantification of influence is reasonable and (b) it can provide useful results, e.g. pointing us to repositories with actual collaboration and overlap at the code level.

c. Finding evidence of significant collaboration. We observe significant collaboration and influence among the repositories in our dataset. First, we identify 28 connected components in our plausible influence graph ($PIT = 0.25$). We find that 71% of the components have less than 5 repositories, while 7% components have more than 15 repositories. In addition, the top 10 most influential repositories have directly influenced 260 repositories in a non-trivial way ($PIT = 0.25$). In fact, the most influential repository has directly influenced 67 repositories.

d. Revealing emerging structures and families. Analyzing the influence graph, we can find interesting lineage and clusters of influence. We find 19 repositories that influenced at least 10 other repositories directly and spawned at least two "families" of repositories. For example, the repository *"vaginessa/android-overlay-malware-example"* is a highly influential information-crawler android malware created on June 17, 2015. It influenced 10 repositories directly and spawned three families of malware: (a) ransomware with 3 repositories, (b) malware for stealing user credentials, such as keyloggers, with 4 repositories, and (c) RAT malware with 3 repositories.

Our work in perspective. Our approach is an essential capability towards understanding the dynamics and evolution of online platforms at the repository level. In fact, it can be seen as a powerful component that can complement other features such as popularity or similarity at the code level, which capture related but different aspects of the repositories.

II. Background

Our work focuses on GitHub, the largest open-source software archive. Here, we provide some background on the repository information that are available in GitHub.

A. The information in GitHub. GitHub is a massive software hosting platform, that enables users to create public repositories to store, share, and collaborate on projects, and provisions a good number of features for the users to do different social networking interactions. The following describes the key elements of a GitHub repository and its author.

1. Repository features: A repository contains the following types of information.

a. Metadata: The most notable metadata fields are: (a) title, (b) description, (c) topics, and (d) readme file. All these fields are optional, and are provided by the repository author. As the

[1] PIMan stands for Plausible Influence Modeling and Analysis.

Symbol	Description
PIMan	Plausible Influence Modeling and Analysis
PIGraph	Plausible Influence Graph
PIScore	Plausible Influence Score between two repositories
PIT	Plausible Influence Threshold to create PIGraph
TPIScore	Total Plausible Influence Score for a repository
RepoSimScore	Code-level Similarity Score between two repositories
RepoPop	Repository Popularity combining number of stars, forks, and watches
RAI	Repository-Author interaction score
AAI	Author-Author interaction score
APop	Author Popularity score in the network
D_All	Dataset with 2089 repositories
D_50	Dataset of 50 repository pairs with $RepoSimScore \geq 0.8$
D_F50	Dataset of random 50 forked pairs of repository
D_3Levels	Dataset of 90 pairs of repositories within 3 ranges of *RepoSimScore*

TABLE I: Table of symbols used in this work.

text fields are provided by the author, they can be most often unstructured, noisy, or missing altogether.

b. Source code: The core element of a software repository is its source code. A repository can contain the software projects which are written in various programming languages such as C/C++, Java, and Python.

2. Social interaction features: It is helpful to group social interactions into repository and author level features.

a. Repository level interaction: GitHub provides functionality for social interaction at the repository level. A repository can be (a) starred, (b) watched to get notification about the updates, (c) receive comments, and (d) forked by other authors.

b. Author level interaction: GitHub enables authors to create a profile by adding social information. Authors can follow other authors which is a direct indication of interest and appreciation. As such, one would expect that followers are likely to be influenced by that author and her repositories.

These two types of interaction define the repository popularity in GitHub which we quantify as *RepoPop*. Note that the repository versus author level features is not that strict; as for example starring a repository by author A implicitly conveys appreciation for both the repository and the author.

B. Fundamental techniques and algorithms. We provide an overview of two fundamental techniques that we leverage in our work: (i) Repo2Vec [2] to represent a repository into a vector and (ii) HackerScope [10] to identify popular authors in GitHub.

1. Code-level similarity - RepoSimScore: Quantifying repository similarity at the code level is not trivial. For validation purposes, we will rely on Repo2Vec [2], an embedding approach that represents a GitHub repository in an M-dimensional vector utilizing data from three types of information sources that enables the repository similarity computation, classification, and clustering tasks. An embedding (a.k.a. distributed representation) is an unsupervised approach for mapping entities, such as words or images, into a relatively low-dimensional space by using a deep neural network on a large training corpus [11], [12]. The approach combines the repository metadata, the code, and the directory structure of the repository to estimate *RepoSimScore*, the similarity between two repositories.

2. Determining node significance in a directed graph: Several approaches exist for capturing the significance of interacting nodes in a complex network. In our case, the interactions are captured by a directed graph, which points to the use of hyperlink-induced topic search algorithm [6] [13]. The algorithm used in these previous studies identifies influential authors by incorporating a HITS approach on the Author-Author graph which captures the interactions of the authors as we will discuss later.

C. Datasets. Our main dataset is D_All, which consists of 2089 Java malware repositories collected by a prior study [2] to whom we are grateful whose goal is to transform a GitHub repository into an M-dimensional embedding vector and determine the similarity between two repositories.

We create three datasets which we use to tune parameters and validate assumptions. First, we create D_50 by randomly selecting 50 pairs of repositories with $RepoSimScore \geq 0.8$. Second, we create D_F50 by randomly selecting 50 pairs of forked repositories. Third, we create dataset D_3Levels as follows. We randomly select 90 pairs of repositories from three ranges of *RepoSimScore*: (a) 30 pairs from range [0-0.25), (b) 30 pairs from range [0.25-0.75), and (c) 30 pairs from range [0.75-1.00].

III. Proposed Method

The main idea behind PIMan is to create a directed weighted graph among repositories by computing Plausible Influence Score, *PIScore*. Our approach can be summarized in three steps. In the first step, we compute the influence score across three dimensions: (a) repository-author interaction, (b) author-author interaction, and (c) author popularity in the network. In the second step, we calculate the Plausible Influence Score (*PIScore*) using the weighted summation of influence scores from these three dimensions. Finally, we create a directed graph where an edge is added between two repositories if their *PIScore* satisfies a defined threshold, *PIT*. The overview of our approach is shown in Figure 2.

Step 1. Computing three influence scores. For a given ordered pair of repositories R1 by author A1 and R2 by author A2, we want to quantify the likely influence of repository R1 on R2. We compute the influence score for the repositories considering three dimensions: R1-A2 interaction, A1-A2 interaction, and popularity score of author A1.

a. Repository-Author interaction: We consider all repository level interactions from author A2 to repository R1, and calculate the influence score for repository-author interaction, *RAI*. First, we compute the Starring Score (SS), Forking Score (FS), Watching Score (WS), and Commenting Score (CS). Here, the Starring Score is equal to 1 ($SS = 1$) if author A2 stars repository R1, otherwise it will be 0 ($SS = 0$). Similarly, we compute (FS), (WS), and (CS) to capture forking, watching, and issue commenting interaction score. Finally, we normalize the score to keep it in the [0,1] range by computing the arithmetic mean of these four scores.

We can combine these scores into a single score using many different ways and by giving different weights to individual

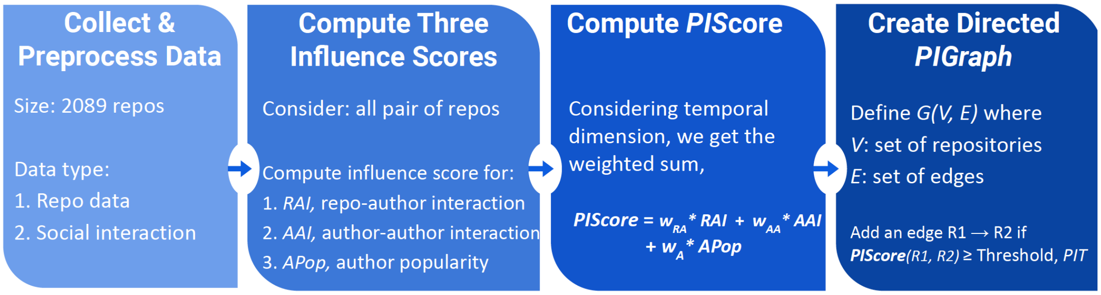

Collect & Preprocess Data
Size: 2089 repos
Data type:
1. Repo data
2. Social interaction
Compute Three Influence Scores
Consider: all pair of repos
Compute influence score for:
1. RAI, repo-author interaction
2. AAI, author-author interaction
3. APop, author popularity
Compute PIScore
Considering temporal dimension, we get the weighted sum,
PIScore = w_RA * RAI + w_AA * AAI + w_A * APop
Create Directed PIGraph
Define G(V, E) where
V: set of repositories
E: set of edges
Add an edge R1 → R2 if
PIScore(R1, R2) ≥ Threshold, PIT

Fig. 2: The overview of PIMan: (a) we define and collect a dataset, (b) for all pairs of repositories in the dataset D_All, we compute three influence scores *RAI*, *AAI* and *APop* from repository-author relationship, author-author relationship, and author popularity in the network, (c) we combine three influence scores to get the Plausible Influence Score (*PIScore*) for each pair, and (c) we create the directed influence graph among the repositories using the *PIScore* value.

scores. Here we decide to first explore using equal weights to all scores, therefore using the following formula:

$$RAI = \frac{SS + FS + WS + CS}{4} \tag{1}$$

As we will see later, this way of calculating the score gives convincing results. In the future, we will explore the use of weights and other ways to combine the individual scores.

b. Author-Author interaction: We consider significant interactions from author A2 to author A1 to calculate an influence score (*AAI*) based on the author-author interactions. First, we compute the Following Score (FS), Other Repository Fork Score (FS_{O_R}), Other Repository Star Score (SS_{O_R}), Other Repository Watch Score (WS_{O_R}), Other Repository Comment Score (CS_{O_R}) if author A2 follows author A1, A2 forks, stars, watches any repository of A1 (except R1), and A2 comments on any repository of A1 (except R1), respectively. Finally, we assign the normalized mean influence score to *AAI* for the aforementioned interactions.

As above, we combine the individual scores giving the same importance using the formula below:

$$AAI = \frac{FS + FS_{O_R} + SS_{O_R} + WS_{O_R} + CS_{O_R}}{5} \tag{2}$$

In future, we intend to consider other ways to combine these interactions.

c. Author popularity: In GitHub, popular authors get more attention and are more likely to influence other authors. The prominence of an author here can be captured by several aggregate metrics such as the number of followers, the total number of stars across all their repositories, etc. as we described previously. As a result, quantifying the overall prominence of an author is not trivial. In order to compute the score, we extend the approach that we mentioned earlier in Section II.

c.1. Generating the author-author interaction graph. We create a graph to capture the network-wide interaction among authors. In more detail, we define a weighted labeled multi-digraph where the nodes are the authors, and we consider six types of relationships that are represented by directed edges with different labels (u, v) from author u to v. These edges can be: (a) a follower edge: when u follows v, (b) a fork edge: when u forks a repository of v, (c) a star edge: when u stars a repository of v, (d) a watch edge: when u watches a repository of v, (e) a contribution edge: when u contributes code in a repository of v, and (f) a comment edge: when u raise a issue comment in a repository of v. These relationships capture the most substantial author-level interactions.

c.2. Edge weight calibration. The above multigraph consists of six different relationships, whose "significance" as an interaction differs. For example, it is "cheaper" to star a repository compared to forking it, which shows intention to use and modify the original code. We want to appropriately weigh the importance of each relationships and, to do this, we consider how rare each relationship is. Intuitively, a rare relationship should get higher importance. Specifically, we consider the weight of a type of edge inversely proportional to a measure of its relative frequency. We calculate the average degree d_{type} for each type of edge, and normalize it dividing by the minimum average degree d_{min}. We get the weight for each type of edge following the equation $w_{type} = \frac{d_{min}}{d_{type}}$: (i) follower edge weight, $w_{follower} = \frac{d_{min}}{d_{follower}}$, (ii) fork edge weight, $w_{fork} = \frac{d_{min}}{d_{fork}}$, (iii) star edge weight, $w_{star} = \frac{d_{min}}{d_{star}}$, (iv) watch edge weight, $w_{watch} = \frac{d_{min}}{d_{watch}}$, (v) contribution edge weight, $w_{contribution} = \frac{d_{min}}{d_{contribution}}$, and (vi) comment edge weight, $w_{comment} = \frac{d_{min}}{d_{comment}}$

c.3. Author popularity score computation. We define two roles of an author in the ecosystem: (a) **producer**, who creates repositories, and (b) **connector**, who interacts with the other authors by following them, and starring, forking, watching, and commenting on their repositories. To quantify the popularity of the author depending on the roles played, we associate each node u with two values: (a) producer score, PS_u, and (b) connector score, CS_u. The algorithm iteratively updates the producer and connector scores until (i) they converge, or (ii) tolerance threshold is reached. First, we initialize PS_u and CS_u to value 1.0. Second, we iteratively update values as follows: (i) for all nodes v with a directed edge to u, (v, u): $PS_u = \sum_v w(v, u) * CS_v$, (ii) for all nodes z with a directed edge from u, (u, z): for all nodes z pointed by u: $CS_u = \sum_z w(u, z) * PS_z$, (iii) we normalize PS_u and CS_u, so that $\sum_u PS_u + \sum_u CS_u = 1$.

We repeat this step until the values converge. For convergence, we set a tolerance threshold for the change of the value of any node. These two scores PS_u and CS_u capture different aspect of authors popularity. Hence, we will get the combined network-wide author popularity score as follows, $APop = PS_u + CS_u$, while in the future we will consider other ways to combine these two scores.

Step 2. Plausible influence score (*PIScore*). We attribute the influence score of repository R1 to repository R2 as *PIScore* $(R1, R2)$.

a. Combining the influence scores. We define the *PIScore* of $R1$ to $R2$ to be the weighted sum of the three scores from repository-author interaction, author-author interaction, and author popularity score. We compute *PIScore* using the following equation,

$$PIScore = w_{RA} * RAI + w_{AA} * AAI + w_A * APop \quad (3)$$

where w_{RA}, w_{AA} and w_A are the weights for the score derived from the repository-author interaction, author-author interaction, and author popularity. We discuss in detail how we calibrate these weights in the next section.

b. Temporal considerations. In general, we propose to adjust the influence score by considering other practical considerations. The most critical is time. The key idea is simple: a recent repository cannot have influenced a repository in the past. However, the implementation can hide several subtleties. We outline two approaches.

Approach 1. We can simply consider the creation time of a repository as a sufficient indication for creating a temporal order. In this approach, if the creation time of repository R2 (T2) is earlier than that of repository R1 (T1), the plausible influence score (*PIScore*) of R1 to R2 is set to zero. Otherwise, we use the influence score as calculated above.

Approach 2. We can consider a "temporal phases of influence" where we recognize that: (a) repositories are created over time, (b) the effect of time can be a real value between zero or one. In other words, we can have a multi-step weight where for different time differences of the repository creation $DT_{creation} = T1 - T2$ we can have different values for a modifying factor **Temporal Modifying Factor (**TMF**)** within [0,1]. For example: one rule could be: if $DT_{creation} > -2$ weeks, then $TMF = 1$, which means we "allow" a repository to influence the repository that was created 2 weeks earlier. The rationale is that software development takes time. Another thought is to consider that a really old repository is less likely to influence a recent repository, say 8 years later, given the fast pace of evolution in software and techniques, so if $DT_{creation} > 8$ years, then $TMF = 0.2$. We can then adjust the *PIScore* by multiplying it with TMF.

Given time and space constraints, we adopted approach 1 in our work, which seems to give meaningful results. In the future, we intend to develop a sophisticated temporal consideration framework. However, such a framework will need to be grounded on observed properties of repositories, such typical duration, temporal properties of the intensity of development as seen by the commits in the code, and observations on how authors interact with other repositories, e.g. how often does an author stars a 8-year-old repository.

Step 3. Creating the PIGraph: We create the PIGraph as a directed weighted graph that captures the plausible influence among repositories. Formally, we define a directed weighted graph: $PIGraph(V, E)$, where V is the repository set, E is the set of edges, and we denote the weight of an edge e as $w(e)$. We consider an edge e between repositories $R1$ to $R2$, if $PIScore(R1, R2)$ (the influence score of $R1$ to $R2$) is greater than or equal to a threshold *PIT*, and assign the weight of the edge $w(e) = PIScore(R1, R2)$.

After generating the PIGraph, each node can be an influencer (having outgoing edges), an influencee (having incoming edges), or both. We use the term **influence outdegree** to refer to the number of outgoing edges of a node. We also define **Total Plausible Influence score,** *TPIScore*, of a repository to be the sum of the weights of all its outgoing edges. Both these metrics capture the network-wide influence of a node as we discuss later.

IV. Tuning and Evaluating Our Approach

We present a systematic approach to select appropriate values for the weight parameters of our approach and we evaluate the effectiveness of PIMan.

A. Tuning the author popularity parameters. As we saw in Section III, the author network consists of six different relationships which show significantly different distribution. To provide appropriate importance, we make the weight of a type of edge inversely proportional to the measure of its relative frequency. This way, we set (i) following edge weight: $w_{follower} = 0.30$, (ii) star edge weight: $w_{star} = 0.48$, (iii) fork edge weight: $w_{fork} = 0.15$, (iv) watch edge weight: $w_{watch} = 0.29$, (v) contributor edge weight: $w_{contributor} = 0.8$, and (vi) comment edge weight: $w_{comment} = 1.0$.

Furthermore, we calculate the popularity of authors, *APop*, by combining the Producer Score (PS) and Connector Score (CS). The algorithm iteratively updates PS and CS for each node in the network. For the convergence, we set a tolerance threshold of 10^{-10} for the change of the value of any node. After 522 iterations, we obtain the converged values of PS and CS for each author. Finally, we assign the sum of PS and CS as the author popularity (*APop*) and rank the authors based on the derived popularity scores.

B. Tuning the weights for the *PIScore* computation. Here, we explain how we can systematically determine appropriate weights to ensure that each type of influence is considered adequately in Equation 3.

a. Weight selection: We choose the weight for each type of influence by "training" the weights to reflect the likelihood that there is code-level similarity between the repositories. Specifically, we use a set of repositories for which we have *RepoSimScore*, the code-level similarity, as we discussed in Section II. We use the Spearman correlation coefficient [14] between the influence score of each dimension and code-level similarity. Note that we only do this once to calibrate the weights.

In more detail, we calculate the weights in two steps by using our D_50 dataset. First, we calculate the correlation coefficient between *RepoSimScore* and influence score of each dimension for each pair of repository in D_50 dataset. We find that *RAI* (Influence Scores of repository-author interaction) is positively correlated to the *RepoSimScore*

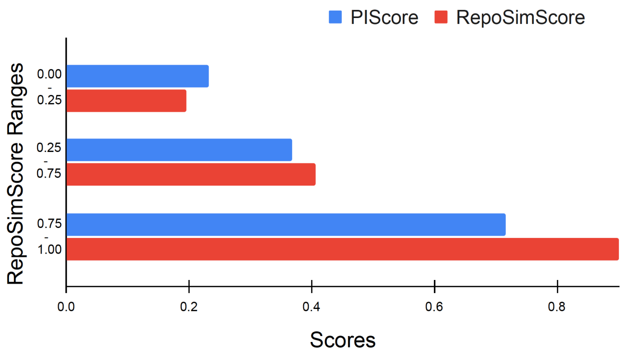


Fig. 3: The Plausible Influence Score (*PIScore*) is highly correlated with the code-level similarity (*RepoSimScore*) (Spearman coefficient, $\rho = 0.79$, and p-value = 1.26e-19) using the D_3Levels dataset.

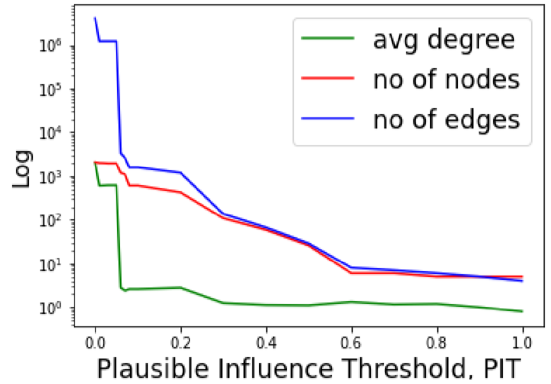


Fig. 4: Increasing the *PIT* threshold reduces the PIGraph network size keeping only higher influence edges.

(Spearman coefficient, $\rho_{RA} = 0.38$ and p-value = 3.82e-9). By contrast, the correlation coefficient of Influence Scores of author-author interaction (*AAI*) and author popularity (*APop*) to the *RepoSimScore* are $\rho_{AA} = 0.15$ with p-value = 1.53e-12 and $\rho_A = 0.06$ with p-value = 1.92e-8 respectively. Finally, we measure the weights in a way that reflects the ratio of the corresponding ρ values, while the sum of the three weights should be equal to one, which leads us to the following weights: $w_{RA} = 0.65, w_{AA} = 0.25$ and $w_A = 0.10$, which we use in Equation 3.

b. Validating our weight selection. We further evaluate the effectiveness of the weight selection of Equation 3 with dataset D_F50 which consists of pairs of forked repositories. The assumption is that forked repositories are supposed to be highly influenced by the original repositories as we discussed earlier. We find that the repository-author interaction score (*RAI*) is the most relevant dimension as the Spearman correlation coefficient with respect to *RepoSimScore* is $\rho = 0.52$ with p-value = 3.32e-11, whereas the values for *AAI* and *APop* are $\rho = 0.22$ with p-value = 1.46e-8 and $\rho = 0.09$ with p-value = 2.73e-8 respectively. These coefficient scores validate that the repository-author relationship is the most relevant dimension in identifying influence among repositories which is why we correctly consider it with a higher weight in our approach as we describe above.

C. Evaluating our approach: We present our effort to establish whether our influence metric provide reasonable results.

Plausible Influence and code-level similarity. We find that our definition of influence correlates relatively strongly with code-level similarity as shown in Figure 3. In more detail, we use the dataset D_3Levels, where each level in that group corresponds to low, medium, and high *RepoSimScore* as explained earlier. (Note that dataset D_3Levels is different than D_50, which we used earlier to determine the weights.) We then calculate the influence score between every pair of repositories in D_3Levels. We plot the average influence score per pair and *RepoSimScore* per pair grouped by level for ease of viewing. We see that the two scores are strongly correlated. Using the original data points, we find a Spearman coefficient $\rho = 0.79$ with p-value = 1.26e-19, which indicates a robust correlation.

To investigate further, we manually assess 10 randomly selected pairs with high influence scores. For example, we find that *"androidtrojan1/android_trojan"* and *"vaginessa/android-overlay-malware-example"* have high *RepoSimScore*. We also find that author *"vaginessa"* follows, stars and forks 5 of repositories of author *"androidtrojan1"*, which leads to a high influence score.

V. Study: Results and Insights

In this section, we will provide additional indications that our approach provides interesting and meaningful results. Specifically, we apply our method on the D_All dataset which provides several interesting observations.

A. Part 1. Studying the PIGraph

A. The effect of the influence threshold *PIT* on the PIGraph. We create the directed influence graph among the repositories in the dataset following the steps described in Section III. We add an edge from repository $R1$ to repository $R2$ if $PIScore(R1, R2) \geq PIT$. We study the PIGraph for different threshold values and plot the graph properties: average degrees, number of nodes, and number of edges in Figure 4. It implies that increasing the threshold reduces the size of the network. In addition, it also exhibits the highly influential characteristics when the threshold $PIT \geq 0.40$. Figure 1 shows that our model produces (a) the dense PIGraph having the threshold $PIT \geq 0.25$ with 426 nodes and 1191 edges, and (b) the sparse PIGraph having the threshold $PIT \geq 0.7$ with 6 nodes and 7 edges.

Observation: The above plot in Figure 4 provides some guidance for selecting a value for the threshold parameter *PIT*. We observe that the distribution of the graph properties creates a knee in the range between 0.1 and 0.35. In order to ensure non-trivial influence, we use a value of 0.25 in the rest of our work unless otherwise stated.

Indirect validation: We argue that this analysis suggests that our influence metric captures a reasonable breadth of behaviors contingent on the *PIT* threshold. Capturing a breadth of behaviors is a desirable property for a modeling approach.

B. The distribution of repository influence. The number of directly influenced repositories follows a skewed distribution with several extremely influential repositories. Here, we focus

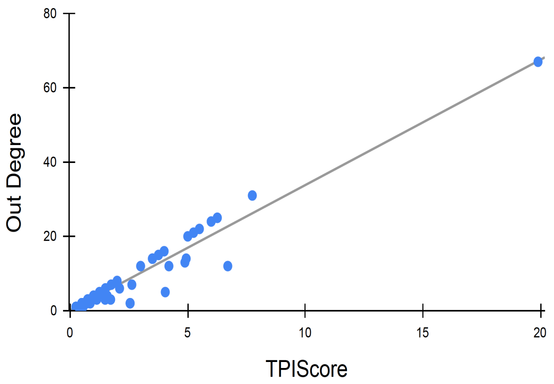


Fig. 5: Number of directly influenced repositories (Outdegree) vs Total Plausible Influence (*TPIScore*) exhibits a linear correlation for D_50 dataset.

No	Influential repositories using PIMan	Popular repositories using *RepoPop*
1	00aj99/AndroidMalware-Example	tiagorlampert/sAINT
2	CCrashBandicot/Android_trojan	adonespitogo/AdoBot
3	CCrashBandicot/Android-KeyLogger	M1Dr05/IsTheApp
4	molotof/sAINT	tomgersic/AndroidKey-Logger
5	5l1v3r1/AndroidRansom-Ware	Mandyonze/Droid-Sentinel
6	CristianTuretta/MAD-Spy	PanagiotisDrakatos/Java-Ransomware
7	tiagorlampert/sAINT	harshalbenake/Android-Elite-Virus
8	Mandyonze/Droid-Sentinel	moloch–/Yoshimi-Botnet
9	androidtrojan1/Android_trojan	androidtrojan1/Android_trojan
10	un4ckn0wl3z/Psyber-Project	siberas/Sjet

TABLE II: Top 10 influential repositories identified by PIMan and popularity metric *RepoPop* (which combines stars, forks, and watches) in our D_All.

on studying the PIGraph ($PIT = 0.25$) and we focus on the edges that represent direct influence: we use the term **influence outdegree** of a repository to refer to the number of directly influenced repositories for a given PIGraph. We find 39% of the repositories having zero direct influence on other repositories while 8% of the repositories influenced at least 20 repositories. In aggregate, the top 10 most influential repositories have directly influenced 260 repositories in a non-trivial way. Furthermore, the most influential repository has directly influenced 67 repositories.

C. Influence: intensity versus the number of repositories. The Total Plausible Influence score (*TPIScore*) provides a different way to capture influence by also considering the intensity of influence. We explore the relationships between *TPIScore* and number of directly influenced repositories (outdegree) by producing the scatterplot shown in Figure 5. There is a strong, arguably linear, correlation between the two metrics. In addition, this plot can help us identify "niche" repositories with a "cult" following: repositories with relatively small outdegree but high influence. As an example, repository *"tiagorlampert/sAINT"* is highly influential (*TPIScore* = 6.71) with only 12 influencees.

B. Part 2. Clusters and Lineage of Influence

A. Finding evidence of collaboration. We want to identify the relationships and groups of high influence. Overall, we observe significant collaboration and influence among the repositories in our dataset. First, we identify 28 connected components in PIGraph ($PIT = 0.25$). We find that 71% of the components have less than 5 repositories while 7% of components have more than 15 repositories. This is a strong indication of substantial collaboration among the repositories, especially if we consider that we have already set a high threshold for the influence in the graph.

Indirect validation: How cohesive are these components? To answer this question, we manually analyze a set of components selected randomly. We find one component with 16 repositories exclusively focused on Android malware, while another component with 235 repositories contained three different families of malware. We argue that this is an additional indication that our approach provides meaningful results.

B. Lineage: highly influential repositories spawn multiple repository "families". In our analysis, we investigate the effect of highly influential repositories, and observe the following phenomenon. We find 19 repositories that have influenced at least 10 repositories directly, and have spawned at least two malware "families". For example, the repository *vaginessa/android-overlay-malware-example* is a highly influential information crawler Android malware created on June 17, 2015. It influences 10 repositories directly and has spawned three families of malware: (a) user credential stealing malware, (b) ransomware malware, and (c) remote access trojan (RAT) malware.

C. Part 3. Repository Influence and Popularity

We compare the repository influence computed by PIMan and GitHub popularity metrics, *RepoPop*. First, we create the PIGraph for our dataset with a reasonable influence score, $PIT = 0.25$. This creates an influence graph with 426 nodes and 1191 edges. We calculate *TPIScore*, the total influence score considering the outgoing edge weights for all nodes, and rank the repositories. Second, we rank the repositories based on *RepoPop*, the popularity metrics of GitHub. We compute the sum of: (i) number of stars, (ii) number of forks, and (iii) number of watches to determine the total popularity score for a repository.

Influential repositories by *TPIScore* versus those ranked by repository popularity. We want to understand the relationship between influence and popularity of repositories. We show the top 10 repositories identified by both approaches in Table II. We find that the top 10 most influential repositories have influenced 260 repositories when the influence is substantial, $PIT = 0.25$. Comparing the two lists in the table, we see that the two approaches have identified different sets of repositories. They have only three repositories in common, and they also differ in their ranking ([7,8,9] versus [1,5,9]). This indicates that the concept of influence captures a different perspective than popularity. That is not to say that one is better than the other: the two concepts are related but not identical.

VI. Related Works

There are relatively few efforts that focus on establishing influence between repositories, especially at the "social" level that we consider here. We discuss the related works below grouped in broad areas of focus.

A. Studies of author level roles in GitHub. The efforts in this group have focused on identifying influential authors and not repositories, as we do here. There are some works [15], [16] that study the ecosystem of developers to measure the social-coding collaboration in GitHub. Focusing on popularity at the author level, some efforts [17], [18] have surveyed developers to study influential users to understand how normal users are influenced by highly influential users on GitHub. Another effort [19] has proposed a ranking-based approach to identify influential authors. A recent work [9] has proposed a Following-Star-Fork-Activity based approach to measure user influence in the GitHub developer social network. A more recent work [6] studies influential authors in a hacker ecosystem in GitHub.

B. Studies on repository popularity. Most prior efforts focus on quantifying and predicting repository popularity, which is not exactly the same as influence. A recent work [7] has proposed an approach to predict repository popularity using starring and following relationships. There are efforts [20], [21] who have used PageRank to identify popular repositories by analyzing the social coding interaction graph, where two nodes are connected. Another work [22] uses network centrality measures to identify influence among Python language repositories. Another effort [8] has studied repository influence, but focused *only* on starring relationships, which ignores many other interactions.

None of these works have addressed the problem as formulated here and in the comprehensive fashion of all the relationships that we use in our work. Our work focuses on establishing influence in the repository level considering (i) repository-author interaction, (b) author-author interaction, and (iii) author popularity in the network.

VII. Conclusions

We present PIMan, a comprehensive approach to establish plausible influence among a set of repositories by capturing social interactions among them. Once we determine the pairwise influence score, we can create a flexible and powerful representation of influence (PIGraph). We showcase the capabilities of our approach by identifying interesting lineage relationships and repository groups rendering significant influence among them. The intention is to highlight the great potential for useful and insightful analysis that our approach can enable.

In the future, we plan to expand the work as follows. First, we will study the relationship between influence and code-level similarity. Second, we will expand our analysis to other types of software: (a) we will compare benign software development with malware software and (b) we will analyze in depth focused software branches, e.g. data mining software, Android apps, etc. Third, we will closely study the malware ecosystem on GitHub as this could provide significant information in combating cyber-crime.

Finally, we intend to open-source our code and datasets to maximize the impact of our work and facilitate follow up research.

Acknowledgements

This work was supported by the NSF SaTC Grant No. 2132642.

References

[1] A. Mockus, D. Spinellis, Z. Kotti, and G. J. Dusing, "A complete set of related git repositories identified via community detection approaches based on shared commits," in *MSR*, 2020.

[2] M. O. F. Rokon, P. Yan, R. Islam, and M. Faloutsos, "Repo2vec: A comprehensive embedding approach for determining repository similarity," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 355–365.

[3] D. Spinellis, Z. Kotti, and A. Mockus, "A dataset for github repository deduplication: Extended description," in *MSR*, 2020.

[4] M. Gharehyazie, B. Ray, and V. Filkov, "Some from here, some from there: Cross-project code reuse in github," in *MSR*. IEEE, 2017, pp. 291–301.

[5] M. O. F. Rokon, R. Islam, A. Darki, E. E. Papalexakis, and M. Faloutsos, "Sourcefinder: Finding malware source-code from publicly available repositories in github," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, 2020, pp. 149–163.

[6] R. Islam, M. O. F. Rokon, A. Darki, and M. Faloutsos, "Hackerscope: The dynamics of a massive hacker online ecosystem," *Social Network Analysis and Mining*, vol. 11, no. 1, pp. 1–12, 2021.

[7] L. Ren, S. Shan, X. Xu, and Y. Liu, "Starin: An approach to predict the popularity of github repository," in *International Conference of Pioneering Computer Scientists, Engineers and Educators*. Springer, 2020, pp. 258–273.

[8] Y. Hu, J. Zhang, X. Bai, S. Yu, and Z. Yang, "Influence analysis of github repositories," *SpringerPlus*, vol. 5, no. 1, pp. 1–19, 2016.

[9] Y. Hu, S. Wang, Y. Ren, and K.-K. R. Choo, "User influence analysis for github developer social networks," *Expert Systems with Applications*, vol. 108, pp. 108–118, 2018.

[10] R. Islam, M. O. F. Rokon, A. Darki, and M. Faloutsos, "Hackerscope: The dynamics of a massive hacker online ecosystem," in *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2020, pp. 361–368.

[11] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, 2014, pp. 1188–1196.

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, pp. 3111–3119, 2013.

[13] L. Li, Y. Shang, and W. Zhang, "Improvement of hits-based algorithms on web documents," in *Proceedings of the 11th international conference on World Wide Web*, 2002, pp. 527–535.

[14] C. Spearman, "The proof and measurement of association between two things." *Appleton-Century-Crofts*, 1961.

[15] A. Lima, L. Rossi, and M. Musolesi, "Coding together at scale: Github as a collaborative social network," in *Eighth international AAAI conference on weblogs and social media*, 2014.

[16] D. Celińska, "Coding together in a social network: collaboration among github users," in *Proceedings of the 9th International Conference on Social Media and Society*, 2018, pp. 31–40.

[17] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian, "Understanding the popular users: Following, affiliation influence and leadership on github," *Information and Software Technology*, vol. 70, pp. 30–39, 2016.

[18] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang, "Why and how developers fork what from whom in github," *Empirical Software Engineering*, vol. 22, no. 1, pp. 547–578, 2017.

[19] Z. Liao, H. Jin, Y. Li, B. Zhao, J. Wu, and S. Liu, "Devrank: Mining influential developers in github," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[20] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network structure of social coding in github," in *2013 17th European conference on software maintenance and reengineering*. IEEE, 2013, pp. 323–326.

[21] R. Bana and A. Arora, "Influence indexing of developers, repositories, technologies and programming languages on social coding community github," in *2018 Eleventh International Conference on Contemporary Computing (IC3)*. IEEE, 2018, pp. 1–6.

[22] W. Ma, L. Chen, Y. Zhou, and B. Xu, "What are the dominant projects in the github python ecosystem?" in *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*. IEEE, 2016, pp. 87–95.