

UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Methods and Tools for Mining and Modeling GitHub Users and Repositories

Permalink

<https://escholarship.org/uc/item/8w80590k>

ISBN

9798180113405

Author

Masud, Md Rayhanul

Publication Date

2026-05-22

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at <https://creativecommons.org/licenses/by/4.0/>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Methods and Tools for Mining and Modeling GitHub Users and Repositories

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Md Rayhanul Masud

June 2026

Dissertation Committee:

Michalis Faloutsos, Chairperson
Nael Abu-Ghazaleh
Chengyu Song
Qian Zhang

Copyright by
Md Rayhanul Masud
2026

The Dissertation of Md Rayhanul Masud is approved:

Committee Chairperson

University of California, Riverside

ACKNOWLEDGMENTS

First and foremost, I owe my deepest gratitude to the Almighty Creator, who has made things work for me at every step of this journey. Every opportunity, every door that opened, and every difficulty that turned into a lesson has come by His grace.

Next, I am deeply indebted to my advisor, Prof. Michalis Faloutsos. Without his support, this journey would not have been possible. He has always inspired me and tried to bring out the best in me; through his guidance I have become noticeably more confident in shaping and presenting my ideas. I will carry the way of thinking he has taught me for the rest of my career.

I am deeply thankful to my dissertation committee, Prof. Nael Abu-Ghazaleh, Prof. Chengyu Song, and Prof. Qian Zhang, for their time, careful reading, and generous feedback. I owe a special thanks to Prof. Qian Zhang for her continuous inspiration as a collaborator; conversations with her have made several parts of this work substantially sharper.

I cannot put into words my gratitude to my wife, Shefa. She has been my constant support throughout these years. Living abroad and pursuing a Ph.D. can be an enormous and painstaking experience, but she never let me feel its weight. She has always inspired me to grow more, and she continues to inspire me every day.

I am profoundly grateful to my parents, and especially to my sister and to my uncles, who took on the responsibility of managing and supporting our family matters back home. Their quiet, steady work freed me to focus on this research; it has been an enormous help that I can only begin to acknowledge here.

I want to specifically thank Rokon, Monir, Risul, and Nishat, who have been by my side throughout. They were at my 24/7 service for any problem (technical, professional, or personal), and I have leaned on them more often than I can count.

I am grateful to my labmates and collaborators in the Faloutsos research group. In particular, I would like to thank Treves Duo (Ben-Maya), Arman, Mimon, Jaminur, Jakapun, Viyom, Vivek, and Sri Shaila G. Working alongside each of them has been a wonderful experience, and the lab would not have been the same place without them.

Beyond the lab, my school friends, university friends, and brothers have been a constant source of encouragement, always reminding me to keep working towards a bright future. I did not, and could not, walk this path alone; I stand here because of all of these well-wishers, and I am deeply grateful to every one of them.

To my beloved wife, Shefa
whose love and support carried me through every step of this journey.

ABSTRACT OF THE DISSERTATION

Methods and Tools for Mining and Modeling GitHub Users and Repositories

by

Md Rayhanul Masud

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2026
Michalis Faloutsos, Chairperson

How can we mine the socio-technical activities of an open-source platform like GitHub? Specifically, how can we (a) identify repositories of interest, (b) study a community of interest at scale, and (c) connect users of interest beyond the platform’s boundaries? GitHub is the largest public software-hosting platform, with over 180 million developers and 630 million repositories as of 2025, yet the research community still lacks general-purpose capabilities to navigate its socio-technical surface end-to-end. The public surface of GitHub exposes far more actionable information than its native search and APIs reach, but extracting it requires methods that scale to the platform. This thesis spans three interrelated problems.

First, we propose RepoScope, a multi-granularity query-by-example framework for GitHub repository search. RepoScope introduces three pre-crawled levels of granularity (292M repositories with minimal metadata, 1.46M with full metadata, and 50.5K with metadata and source code) and combines them with an embedding-based retrieval pipeline and a clustering layer with interpretable summaries. RepoScope outperforms the GitHub REST

API on 70% of queries and matches or exceeds it on 94% of cases, achieving an average top-10 precision of 86% and a 100% success rate on its largest dataset.

Second, we propose MAGNET, a systematic framework for studying malicious-intent malware repositories on GitHub as a software ecosystem. MAGNET integrates three stages: (a) an LLM-agent pipeline that identifies malicious-intent repositories by combining repository-level evidence with the author’s portfolio, (b) a three-level taxonomy of engineering archetypes, and (c) three participation roles (co-developer, supporter, follower) that model how users engage. MAGNET identifies 3,598 malicious-intent repositories from 16,726 candidates with 97% human-validated precision, shows that 80.24% are ephemeral one-shot uploads, and uncovers an active community of approximately 51K users across 711 repositories in which following dominates co-development.

Third, we propose GeekMAN, a human-inspired approach to identify similar usernames across online platforms with a focus on *technogeek* users on security forums and GitHub. GeekMAN integrates (a) decomposing usernames into chunks, (b) de-obfuscating slang conventions, and (c) exhaustively comparing the resulting decompositions to score similarity. On technogeek datasets spanning 1.8M usernames from GitHub and five hacker-rich security forums, GeekMAN matches with 81–86% precision, against 46–78% for two state-of-the-art baselines, and identifies 6,327 security-forum users that match malware authors on GitHub. Together, these contributions form a systematic suite of methods and tools for mining the socio-technical activities of an open-source platform and enable studies that were previously out of reach.

TABLE OF CONTENTS

List of Figures	xi
List of Tables	xiv
1 Introduction	1
2 RepoScope: A Multi-Granularity Query-by-Example Framework for GitHub Repository Search	8
2.1 Background	13
2.1.1 Motivating Example.	13
2.1.2 GitHub Search and Limitations.	14
2.1.3 The Searching Challenges Revisited.	15
2.2 Proposed Method	17
2.2.1 Multi-Granularity Data Collection.	18
2.2.2 Repository Embedding and Retrieval.	19
2.2.3 Result Interpretation and Context.	22
2.3 Experiments and Evaluation	24
2.3.1 RQ1: RepoScope vs. the GitHub API.	29
2.3.2 RQ2: Efficacy at Different Levels of Granularity.	30
2.3.3 RQ3: Efficacy at Different Result Size TopK.	31
2.3.4 RQ4: Result Clustering and Summarization.	31
2.3.5 User Study to Evaluate RepoScope.	32
2.3.6 Search Latency and Storage Requirement.	34
2.3.7 Evaluating ChatGPT in Labeling Similar Repositories.	35
2.4 Discussion	36
2.5 Threats to Validity	38
2.6 Related Work	39
2.6.1 Platforms and Datasets.	39
2.6.2 Establishing Similarity Between Software Repositories.	40
2.7 Conclusions	41

3	MAGNET: A Software-Engineering View of the Malware Ecosystem on GitHub	42
3.1	Identifying Malicious-Intent Repositories	47
3.1.1	Definition	48
3.1.2	Candidate corpus and evidence	48
3.1.3	LLM-agent labelling	49
3.1.4	Manual validation	50
3.2	Trends and Growth	50
3.2.1	The growth of the malware ecosystem	51
3.2.2	The growth by malware type	52
3.3	Engineering Archetypes	52
3.3.1	Features and clustering procedure	53
3.3.2	The three-level taxonomy	54
3.3.3	The four engineering modes	55
3.4	User Communities and Participation	56
3.4.1	Three role views	57
3.4.2	Graph construction and communities	57
3.4.3	Findings per role	58
3.4.4	The participation pyramid	59
3.5	Discussion	60
3.6	Related Work	62
3.7	Conclusions	63
4	GeekMAN: Disambiguating Usernames Across Platforms	65
4.1	Background and Data	70
4.2	Quantifying Technogeek Usernames	72
4.3	Proposed Method	75
4.3.1	Our Approach in Detail	76
4.3.2	Computational complexity	82
4.4	Experiments and Evaluation	83
4.4.1	Evaluating GeekMAN	83
4.4.2	Case study: Matching GitHub malware authors	90
4.5	Discussion: Tool and Scope	92
4.6	Related Work	96
4.7	Conclusion	98
5	Conclusions	99
	Bibliography	102

LIST OF FIGURES

2.1	RepoScope identifies and clusters 1K similar GitHub repositories to <i>nehamchangappa/Online-Banking-Web-Application-Project</i> , shown as a 2D projection of a multidimensional space. RepoScope summarizes these clusters, identifying banking and general web applications, and considering Java and other programming languages.	9
2.2	The RepoScope architecture and key functions: (a) RepoScope collects repository data at three granularity levels here: D_All, D_Pop and D_VIP with different sets of features $F1$, $F2$ and $F3$, respectively. (b) RepoScope creates and stores the embeddings of the repositories. Given a query repository and its embedding, RepoScope searches in the database to retrieve the top-k similar repositories from the three datasets, for a user-specified k. (c) Finally, the search results are clustered and summarized to provide context.	17
2.3	Cluster summarization prompt for Gemma3-1B-it.	23
2.4	ChatGPT-4 prompt for labeling a pair of repositories (query, result) based on their functional similarity.	28
2.5	User ratings on using RepoScope for Q2, Q4 and Q5.	32
2.6	Contextual information: RepoScope returns 1K repositories for the query repository <i>davidson679/Bypass-Web-Application-Firewalls</i> and provides meaningful context: (a) clustering hierarchy of the returned repositories as dendrogram, and (b) cluster profiles: summaries and a two dimensional projection of the three clusters that the user selected (dotted vertical line on dendrogram).	34

3.1	Decomposition of the 3,598 malicious-intent repositories into nine engineering archetypes via three orthogonal splits. L1 separates <i>Ephemeral</i> ($N=2,887$, 80.3%) from <i>Sustained</i> ($N=711$, 19.7%) repositories on lifecycle and participation features. L2 : artifact content splits Ephemeral into <i>Source Snapshots</i> and <i>Artifact Drops</i> ; collaboration process splits Sustained into <i>Quiet Iterative</i> and <i>Discussion-Driven</i> . L3 applies the ecosystem-visibility features to all four L2 clusters.	43
3.2	Malicious-intent corpus identification. Starting from 16,726 candidates, two evidence-based filters yield 3,598 high-confidence malicious-intent repositories.	47
3.3	Prompt template used by the LLM agent	48
3.4	Repository creation by three-year window. The corpus grows in multiplicative waves and is concentrated after 2016.	50
3.5	Engagement distribution across the 3,598 malicious-intent repositories. (a) Most repositories are silent on engagement signals: 91.8% have zero pull requests, 86.4% have zero issues, and 83.6% have a single contributor. (b) Commits rises slowly from 0.10 at $x = 1$, indicating the widest spread of values, while issues and pull requests saturate near 1.0 almost immediately, indicating that the great majority of repositories accumulate little or no public coordination.	51
3.6	The three-role decomposition. Each role uses the same repository universe but keeps a different class of repo-user interactions.	57
3.7	Per-role structure of the sustained corpus. (a) Share of the 51,194 distinct users who participate in each role; roles overlap, so shares do not sum to 100%. (b) Venn decomposition: only 1,078 users sit in two or more roles, and just 75 in all three.	60
4.1	GeekMAN calculates the similarity score for a pair of usernames focusing on technogeek users. Our approach tries to emulate human interpretation by combining the (a) Chunkification, (b) Deslangification, and (c) Comparison modules.	68

4.2	Distribution of percentage of population who use digit in between letters multiple times. An example username containing two digits in between letters: <i>g3ntl3man</i>	73
4.3	The CCDF of total number of chunks found per username across different types of online platforms. An example username: <i>sniper7kills</i> having three chunks $\{sniper, 7, kills\}$	74
4.4	The word-cloud that shows popular slangified words used at least 10 times as usernames or as parts of usernames in technogeek forums (200 words) than social media (6 words only).	75
4.5	An example of how the modules of our approach will handle a pair of usernames: <i>z3r0c00l</i> and <i>COOL_zERO</i>	76
4.6	The effect of <i>SimT</i> on the performance of GeekMAN. The blue vertical line at 0.7 marks the point that finds a reasonable trade-off with a slight emphasis on Precision.	85
4.7	GeekMAN Platform: Matching usernames <i>Anon-Exploiter</i> and <i>An0n3xploit3r</i> . The platform is publicly accessible at https://geekman.streamlit.app . . .	94

LIST OF TABLES

2.1	Our three-level database and its two instances (v1 and v2 collected in 2023 and 2024).	13
2.2	Performance evaluation of RepoScope and the GitHub API for top-10 search results on our QuerySet.	26
2.3	Performance evaluation of RepoScope and GitHub REST API in level L1 over different TopK values across randomly selected five queries from the QuerySet.	26
2.4	Data collection and storage for the datasets in RepoScope.	35
2.5	Comparison of RepoScope with other related approaches.	40
3.1	Features used at each level of the taxonomy, grouped by conceptual axis. All features are computed from GitHub metadata after bot filtering.	53
3.2	Level-2 archetypes and signatures.	54
3.3	Sustained-scope role coverage after cleaning. A strict community has at least three users and three edges.	58
4.1	Summary of our datasets: security forums, GitHub, and social media. . . .	71

4.2	Definition of terminology for GeekMAN approach.	77
4.3	Performance comparison analysis for the algorithms in D_All using $SimT=0.7$ for GeekMAN with annotator agreement Kappa Score $K=0.53$	87
4.4	Performance comparison analysis for the algorithms in D_Slang using $SimT=0.7$ for GeekMAN with annotator agreement Kappa Score $K=0.54$	87
4.5	Performance metrics between GeekMAN and the baseline algorithms across social network ground truth.	88
4.6	Real username matches that baseline algorithms missed.	90
4.7	GeekMAN finds more forum-GitHub matches ($SimT=0.7$) compared to exact matches alone. A substantial percentage of them are heavy-hitters in that forum. Note that a malware author can match users in more than one forum.	91

Chapter 1

Introduction

“How can we mine the socio-technical activities of an open-source software platform like GitHub?” Specifically, how can we (a) identify repositories of interest, (b) study a community of interest at scale, and (c) connect users of interest beyond the platform’s boundaries? This question is motivated by a real need: there is a growing gap between the information that public software platforms expose and the capabilities researchers and practitioners have to navigate it systematically. Our work is driven by the following insight: the public surface of GitHub (metadata, source code, issues, pull requests, stars, forks, followers, and the author’s portfolio) is much richer than its native search and APIs let us reach, and a small set of well-designed methods can recover this information end-to-end.

GitHub is the largest open-source platform, hosting more than 180 million developers and 630 million repositories as of 2025 [29, 35]. The mechanisms that make GitHub useful for legitimate software engineering also expose a rich socio-technical fabric that, until recently, has been explored only opportunistically. Surfacing functionally similar repositories at scale

supports learning, code reuse, prototyping, and benchmark construction [25, 76, 119, 15, 60]. Characterizing a community of repositories supports security analysts and platform-governance researchers who need to understand not only *whether* a repository is malicious, but *how* it is developed, maintained, and consumed by its audience [85, 71, 99]. Tracing users across platforms is essential for studies of cybercriminal ecosystems, where the same person often appears on GitHub, security forums, and social media with related but rarely identical usernames [49, 90].

Researchers are limited by the absence of general-purpose tools for these three problems. The native GitHub search is constrained by rate limits, capped result sets, and a black-box ranking that supports only keyword queries [15]. Studies of malicious activity on GitHub have to date focused largely on *which* repositories contain malware, not on the engineering practices and user communities that surround them [85, 10, 82]. Cross-platform user tracing has overwhelmingly been formulated for mainstream social media, where usernames are clean and profile signals are available; neither holds for the obfuscated, multi-chunk “technogeek” usernames that dominate the hacker ecosystem [81, 114, 110].

Therefore, the question that we answer in this thesis is: “How can we mine the socio-technical activities of GitHub at scale?” At a high level, the goal is to develop methods and tools that reveal what software artifacts are being created, who participates around them, and how these actors develop, maintain, discuss, and propagate software through the public surface of GitHub. We study this problem across three connected levels: repositories of interest, communities of interest, and users of interest.

The problem statement can be captured well in the following motivating questions:

Question 1: How can we identify repositories of interest at GitHub scale?

Question 2: How can we study a community of interest as a software ecosystem on GitHub?

Question 3: How can we connect users of interest on other online platforms using usernames only?

In Question 2, the community of interest we study in this thesis is the malicious-intent malware repositories on GitHub, which we treat as a case study of a wider socio-technical phenomenon.

We face several practical challenges that span data acquisition, signal interpretation, evaluation, and decision-making. First, studying GitHub at scale requires reliable access to a platform that is both large and continuously changing. A live-query approach is limited by API quotas, capped result sets, and opaque ranking, which restrict both coverage and reproducibility. A snapshot-based approach can support more repeatable large-scale analysis, but it introduces its own burden: the data must be collected, stored, refreshed, and indexed so that millions of repositories and users can be searched and analyzed efficiently. Second, repositories expose heterogeneous and noisy signals. A single repository may contain short metadata fields, an unstructured README, source code in multiple languages and packaging formats, and social signals such as commits, issues, pull requests, stars, and forks. These signals capture different aspects of software activity: some describe what the artifact is, some reveal how it is maintained, and others show how users react to or propagate it. Therefore, they cannot be collapsed into one generic representation without losing important distinctions.

Third, reliable ground truth is scarce, and each task introduces a different kind of ambiguity. For repository search, there is no canonical set of relevant repositories for a given example, so evaluation requires human judgment over carefully curated query repositories. For malicious-intent repositories, the boundary between malicious and benign is context-dependent because similar code can appear in offensive, educational, research, or defensive settings; therefore, labeling requires both code-level evidence about what the artifact does and author-level evidence about how it is framed and used. For cross-platform user matching, verified identity links are rarely public. This is especially difficult in the username-only setting studied in this thesis, where users on security forums, paste sites, and code-sharing boards often adopt creative handles with digit substitutions, symbol insertions, abbreviations, slang, or multiple embedded words. Such usernames break the assumptions of exact string matching and make mainstream profile-based matching methods difficult to apply.

Fourth, converting these signals into actionable decisions is itself nontrivial. Repository embeddings must be calibrated so that closeness in embedding space corresponds to human notions of repository similarity. User roles around repositories must be separated before studying a community, since a committer, an issue participant, a stargazer, and a forker reflect different forms of participation. Finally, ambiguous and obfuscated usernames require a systematic interpretation procedure that can approximate human reasoning while producing stable and reproducible matches. Together, these challenges motivate the methods developed in this thesis.

There has been limited work on the problems as we formulate them here. Prior GitHub search has either accepted the API’s keyword-only contract [2, 15] or focused on similarity

within small curated sets without addressing platform-wide search [86, 77, 70, 119]. Studies of malicious activity on GitHub have focused on identification and actor profiling [85, 82, 49, 91, 80, 97] or on code-centric properties of malware source [9, 10], leaving the software-engineering view of these ecosystems open. Cross-platform username matching has been developed primarily for mainstream social networks [81, 114, 110, 61, 107, 37, 117, 74, 62, 63, 118], and these methods underperform on the obfuscated, chunked usernames that characterise hacker-rich platforms.

Our work is arguably the first effort to provide a unified, end-to-end tool stack for mining the socio-technical surface of GitHub. The key contributions of this thesis are as follows.

- **RepoScope**, a multi-granularity query-by-example framework for GitHub repository search (Chapter 2). Given a repository of interest, RepoScope returns functionally similar repositories using a three-level pre-crawled database, an embedding-based retrieval pipeline, and a cluster-summarizing layer.
- **MAGNET**, a software-engineering framework for studying malicious-intent malware repositories on GitHub as an ecosystem (Chapter 3). MAGNET combines an LLM-agent identification pipeline, a three-level taxonomy of engineering archetypes, and three behavioral roles capturing user participation.
- **GeekMAN**, a human-inspired approach to identify similar usernames across online platforms with a focus on technogeek users (Chapter 4). GeekMAN integrates chunkification, deslangification, and exhaustive comparison into a single similarity score and, as a use case, connects GitHub malware authors to their footprints in security forums.

Our key results are summarized as follows.

a. RepoScope provides scalable and effective GitHub repository search. Our approach outperforms the GitHub REST API on 70% of queries and matches or exceeds it on 94% of cases. RepoScope achieves average top-10 precision of 86% and 100% success rate on its largest dataset, outperforming the GitHub API by 20 percentage points in precision while operating over 292M public repositories without GitHub’s API limits.

b. MAGNET reveals an ecosystem that is wide but shallow. Starting from 16,726 malware-candidate repositories, MAGNET identifies 3,598 malicious-intent repositories from 3,487 authors with 97% human-validated precision. 80.24% are ephemeral one-shot uploads, while the sustained minority concentrates almost all observable engineering activity. We further find an active community of approximately 51K users across 711 repositories in which following dominates co-development, suggesting that the ecosystem is more visible as an audience surface than as a collaborative development space.

c. GeekMAN outperforms prior username-matching methods on technogeek users. On technogeek datasets, GeekMAN achieves 81–86% precision with 72–75% relative F1, compared with 46–78% precision and 38–57% F1 for two state-of-the-art baselines. As a use case, GeekMAN identifies 6,327 security-forum users that match malware authors on GitHub; our method adds substantial value beyond exact string matches and outperforms prior methods even on mainstream social-network usernames.

We release our datasets, ground truth, and online tools to maximize impact. Follow-up efforts can use our approach to (a) build continually updated query-by-example services for GitHub and adjacent platforms such as GitLab and BitBucket, (b) extend MAGNET

to benign software-engineering ecosystems, and (c) integrate GeekMAN with content-level signals to push cross-platform user tracing further. We believe these methods provide a foundation for future studies of open-source platforms.

The rest of the thesis is organized as follows. Chapter 2 introduces RepoScope and our approach to identifying repositories of interest. Chapter 3 introduces MAGNET and our approach to studying a community of interest at scale. Chapter 4 introduces GeekMAN and our approach to connecting users of interest beyond the boundaries of a single platform. Chapter 5 concludes the thesis and discusses directions for future work.

Chapter 2

RepoScope: A Multi-Granularity Query-by-Example Framework for GitHub Repository Search

How can we enable efficient large-scale exploration of open-source codebases like GitHub?

An effective repository search capability is essential, yet no comprehensive solution currently exists. The existing GitHub API search is limited by: (a) the lack of support for example-based queries, (b) strict API request constraints and caps on result size, and (c) its inherent, black-box criteria that prevent developers from refining searches to their specific needs.

GitHub has become the most popular open-source software platform, hosting more than 420 million private and public repositories as of 2024 [29] with a continued yearly growth [55, 48].

Software practitioners and researchers require a robust search capability to navigate this vast repository ecosystem. Such a capability is essential for various use cases, including

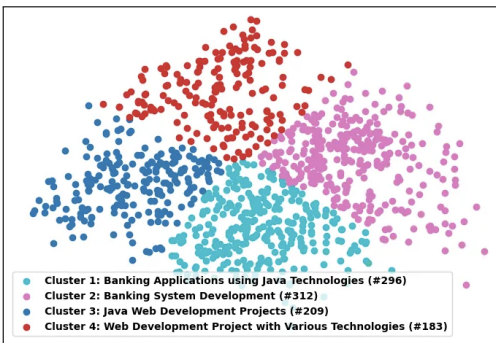


Figure 2.1: RepoScope identifies and clusters 1K similar GitHub repositories to *nehamchangappa/Online-Banking-Web-Application-Project*, shown as a 2D projection of a multidimensional space. RepoScope summarizes these clusters, identifying banking and general web applications, and considering Java and other programming languages.

learning [5], bootstrapping [44], and project reuse [25, 76] as well as advanced research in software engineering [119, 15]. For example, ReposVul [109] mines GitHub repositories to extract a vulnerability dataset that enables researchers to systematically analyze common coding practices and their weaknesses to identify security issues across open-source projects. More recently, the rise of large language models (LLMs) in software engineering [47] has further amplified the demand to identify appropriate repositories for training and evaluation [60].

Problem. We focus on the following problem formulation: given a GitHub repository of interest, our objective is to identify functionally similar repositories on a GitHub-wide scale. The term *similarity* in the scope of repository search can vary, ranging from functional similarities [70, 119] to source code-level comparisons [76, 12, 86]. Moreover, the format of the query can differ significantly, following various user intents, such as keyword-based searches [57], natural language queries [13, 12], and example-based searches [119, 77, 86]. Given a repository of interest, our goal is to enable users to discover software solutions that implement similar functionalities. Recent studies have underscored the significance of

code reuse in software development, revealing that a substantial proportion of projects (up to 80%) reuse existing source code [52] and incorporate third-party components [45]. This highlights the substantial benefits of query-by-example repository search for both developers and researchers.

Challenges and Current Practices. Despite the potential benefits, identifying similar repositories among millions on GitHub is a challenging task. Existing approaches like the GitHub API primarily rely on *keyword-based* searches, which often yield ambiguous results due to minor variations in search terms [15]. Though GitHub’s search scope customization across different features (e.g. description, topics) allows users in utilizing desirable features in search result processing, the GitHub API is limited by a 5K requests per hour constraint and a maximum of 1K results, hindering holistic repository searching. To overcome these API constraints, platforms like GHS (GitHub Search) [15] provide *keyword-based* repository search capabilities, but they lack comprehensive metadata such as topics and readme files, leading to limited coverage. Other efforts like WorldOfCode [65] and GHArchive [24] offer datasets and platforms, but they are not designed for repository search, let alone query-by-example searches. Moreover, there have been other research efforts which develop sophisticated algorithms to establish similarity between software repositories using high-level metadata analysis [119], topic modeling [56, 103], graph-based relationship [77] and embedding techniques [86, 82, 59]. However, these approaches are not designed to support large-scale exploration on GitHub, particularly when considering the vast number of repositories.

This Work. We propose **RepoScope**, a multi-granularity query-by-example framework to search the public repositories of GitHub. RepoScope introduces three distinct levels of granularity, defined by: (a) the number of repositories, and (b) the amount of information per repository at each level. In addition, RepoScope supports: (a) querying by example, and (b) provides hierarchical clustering and cluster profiling information. We discuss the key capabilities of our approach.

- *Multi-granularity Data Collection.* As our first step, we define three levels of granularity and collect the related information, and we do this periodically. Having the data locally allows our search to be free from GitHub’s API restrictions. Our three levels are: (L1) D_All, which includes 292 million repositories with minimal metadata; (L2) D_Pop, comprising 1.46 million relatively popular repositories with complete metadata; and (L3) D_VIP, containing 50.5K highly popular repositories with both metadata and source code. These levels allow the user to balance the number of repositories considered in the search versus the amount of information that is used in establishing the similarity.

- *An Efficient Embedding-based Similarity Search.* For each level of granularity, we determine and calculate the embedding of each repository to incorporate functional similarity and overcome the limitation of keyword-based searching. We use embedding techniques BERT [17] and CodeBERT [22] to generate embeddings based on the available information at each level (as we explain later). Finally, LLM2Vec [6], a dense embedding method, is used to re-rank the search results for better precision. This way, we support efficient query-by-example searches.

- *Result Interpretation and Visualization.* Going beyond a list of repositories, we enrich our responses by providing: (a) clusters of repositories using the embedding presentation, and (b) a high-level summary of each cluster using an LLM-based approach. A 2D visualization of the multidimensional clusters is shown in Figure 2.1 for the target repository. In addition, our approach can reveal the clustering hierarchy within each cluster using a dendrogram approach, as we show in Figure 2.6.

Results. We evaluate the ability of our approach to find similar repositories through a series of comprehensive experiments. We use QuerySet, a curated set of 146 GitHub repositories. We evaluate our approach against the GitHub REST API in terms of: (a) precision in the top-10 results, and (b) success rate, where we define a response as successful if the top-10 returned repositories include at least one similar repository. We employed the ChatGPT API (gpt-4.1) to validate our method and baseline results, with manual checks confirming its accuracy. Our results show that RepoScope outperforms the baseline in 70% of the QuerySet repositories, and achieves at least the same performance in 94% of cases. Our approach demonstrates an average precision of 86% for the D_All dataset achieving a 100% success rate, which surpasses the GitHub API’s 66% precision. Furthermore, RepoScope consistently surpasses GitHub API in every metric across top-10, top-20, top-50, and top-100 search results. For $k=100$, RepoScope achieves an average precision of 73%, while GitHub API achieves 64%. Moreover, RepoScope’s silhouette score of 0.54 and BLANC score of 0.58 indicate sound clustering and summarization.

Table 2.1: Our three-level database and its two instances (v1 and v2 collected in 2023 and 2024).

Level	Dataset	Features	Version	# Repositories	Crawling Date
L1	D_All	$F1 = \{\text{Repo Name, Description}\}$	v1	292M	Sep'2023
			v2	358.9M	Sep'2024
L2	D_Pop	$F2 = F1 + \{\text{Topics, Readme}\}$	v1	1.46M	Sep'2023
			v2	1.61M	Sep'2024
L3	D_VIP	$F3 = F2 + \{\text{source code}\}$	v1	50.5K	Sep'2023
			v2	53.2K	Sep'2024

We will release RepoScope as an online platform as we show in Figure 2.6. It can be used by researchers and practitioners as the go-to capability in exploring the vast number of public repositories, as we discuss further in the next section.

2.1 Background

We first motivate the need for our work and then provide the necessary background context.

2.1.1 Motivating Example.

We present a realistic scenario to highlight the challenges our work addresses. Alison, a data scientist, aims to fine-tune an LLM for an AI-assisted web-crawler. She plans to create a dataset of web-crawler projects from GitHub. She discovers a popular web-crawler GitHub repository, ‘code4craft/webmagic’ [31], and wants to find and collect similar projects to it to better understand the landscape of web crawlers for her training. However, GitHub

lacks a built-in feature for this type of search. Instead, Alison uses keyword-based search to approximate example-based query support. When she uses description of the repository ‘A scalable web crawler framework for Java’ as query, she retrieves 3 relevant repositories. Removing language (*Java*) yields 3 more, and discarding the article ‘A’ again brings 3 more, indicating the limitation of keyword-based searches. However, searching with the keywords ‘web crawler’ shows 27.3K results, but GitHub restricts her to retrieve only 1K results. Moreover, after briefly browsing those 1K results, Alison finds that the returned results are varied in diverse purposes and usages; some are generalized crawlers with advanced features, some are data-sensitive crawlers, others are simple basic implementations, and so on. Thus, she cannot interpret the variety of the result scope in a meaningful way.

Alison faces three challenges: finding similar repositories with example-based queries, querying for semantically relevant repositories without result limits, and interpreting the retrieved results. Our goal is to assist Alison in overcoming these challenges.

2.1.2 GitHub Search and Limitations.

We discuss in detail the features that repositories in GitHub have and how they are used by the GitHub search.

- *Repository Features.* A GitHub repository consists of both the related code and meta-data, namely a rich set of informative features. Specifically, the metadata consists of the author name, the title, a description, topics, and a README file, which we briefly discuss below. A title is supposed to capture the essence of project [88], while descriptions and topics offer additional but brief context [94], including goals, limitations, or technological details. The README file is typically much larger, and explains the functionality of the

project and very often provides a comprehensive guide [64], including setup instructions, usage examples, and contribution guidelines. We elaborate on metadata features in the next section.

- *GitHub Search: keyword-based.* GitHub search provides a fairly restrictive interface with respect to what repository features are used and how they are used in a search. Users can search for repositories using keywords in titles/descriptions/topics/readme contents to find projects aligned with specific interests (e.g. “machine learning”). The user can also define a collection of keywords that should either appear or not be present within the metadata of a repository. However, searching by keywords can miss out on important contextual information as we discussed earlier. Note that users can apply different filtering mechanisms using stars, forks, date, and other attributes, which are orthogonal to the problem we address in this study: they capture aspects, such as popularity of a repository and impact.

2.1.3 The Searching Challenges Revisited.

Current solutions have difficulty addressing the problem as we formulate it here.

- *Query-by-Example Support.* Though users can be greatly benefited from *Query-by-example* based repository search, there is limited attention from the research community in this regard. Query-by-example eliminates the need to manually determine keywords and the need to invent tricks to overcome the keyword-based limitations and the ambiguities.

- *Searching GitHub at scale.* Searching *all* the repositories on GitHub is a challenge. The GitHub search API imposes significant constraints, including a limit of 5K requests

per hour and 1K results per query [14]. The brute force method for circumventing these restrictions is impractical: downloading the entire corpus of GitHub repositories with their metadata and source code, would take over a year and require several petabytes of storage [46], at a rate of 10 repositories/second.

- *Search Scope Customization.* Given a target repository, the scope of a similarity search study can range from broad to narrow utilizing more to fewer features in search result processing, serving the intention of the search. In addition, the search can be: (a) broad, as in exploring the landscape across GitHub by providing clusters of 300K similar repositories, to (b) narrow, as in identifying the top-5 most similar repositories. The scope could be something in-between or even a hybrid: it could start as a broad search and then interactively narrow down to two, three sub-areas, such as the top-5 most repositories that implement the same functionality on different hardware platforms.

- *Semantic Ambiguity.* Searching with keywords introduces uncertainties. A text-book example appeared in a real study [85] that was searching for malware code repositories. Searching with keywords such as *malware*, *DDoS*, *trojan* would also return repositories with countermeasures as in *malware detection* etc. The keyword-based searches lack the semantic and contextual understanding and synonymy handling, which degrades the accuracy of the results.

- *Lack of Contextual Information.* A major limitation of existing approaches is the lack of context accompanying the search results, which hinders users' ability to comprehend the result space. It is not clear how keywords are prioritized, and there is no information on the relative similarity between repositories that could reveal families of related work.

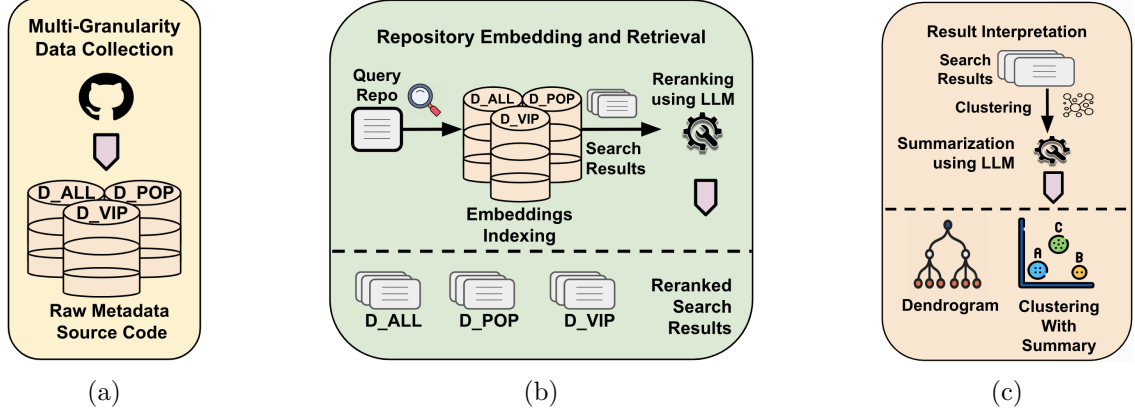


Figure 2.2: The RepoScope architecture and key functions: (a) RepoScope collects repository data at three granularity levels here: D_All, D_Pop and D_VIP with different sets of features $F1$, $F2$ and $F3$, respectively. (b) RepoScope creates and stores the embeddings of the repositories. Given a query repository and its embedding, RepoScope searches in the database to retrieve the top-k similar repositories from the three datasets, for a user-specified k. (c) Finally, the search results are clustered and summarized to provide context.

2.2 Proposed Method

RepoScope utilizes a holistic offline-online search mechanism to enable query-by-example repository search in GitHub at scale. In Figure 2.2, we show three components that work in concert: (a) Multi-Granularity Data Collection, (b) Repository Embedding and Retrieval, and (c) Result Interpretation. The key insight of our approach is its ability to collect public data from the GitHub repository and provide mechanisms to identify similar repositories. Given the GitHub API’s hit rate limits, which can pose a significant bottleneck in repository mining, we employ an offline search by collecting, preprocessing and indexing a set of repository data in advance, to overcome these limitations. Initially, the crawled repositories are represented in the embedding space to overcome the constraints of keyword based search. During the online phase, when a query repository is provided, RepoScope retrieves and ranks similar repositories, followed by clustering and summarizing the search results.

2.2.1 Multi-Granularity Data Collection.

To balance breadth and depth, we generate datasets with varying detail levels. Users searching broadly with minimal info like title and description can quickly find a wide range of repositories. In contrast, for high-quality results, queries should use additional metadata like topics, readme files, and source code. This allows users to explore available projects broadly or obtain a refined, relevant set of high-quality results based on their focus. To address these challenges, we crawl and create three datasets with different levels of granularity. Each level (L_x where $x=1,2,3$) targets a different set of repositories as stated in Table 2.1.

- *Level 1: D_{All} : Name, and Description.* This dataset contains minimal repository information of 292M repositories (in v1 of our dataset); repository name, description.

Collecting the data for all (or nearly all) public repositories in GitHub is not straightforward. We leverage two things: (a) each repository is associated with a numeric identifier ID ; and (b) we can retrieve repositories using consecutive ID numbers. In more detail, we use GitHub API [34] that takes a repository ID and returns the next 1K repositories in increasing ID value: $ID+1$, $ID+2$, ... etc. We start from $ID = 1$, and apply the approach iteratively to gather all the repositories available at the time of crawling.

- *Level 2: D_{Pop} : All Metadata.* For each repository at this level, we collect all its metadata, namely: name, description, topics, and readme file. As we discussed earlier, we can create several datasets at this level using different criteria. We opted to collect *relatively visible* public repositories, which we define as: (a) at least 5 stars, (b) at least 5 forks, (c) not forked from another repository, and (d) with a non-zero size. This combination of criteria is often used to eliminate unused, replicas, and empty repositories in the literature [38, 57].

- *Level 3: D_VIP: Metadata and Code* . Repositories at this level have all their metadata and source code. It is clear that crawling, storing and processing that much data per repository is orders of magnitude demanding compared to the previous two levels [65]. We create D_VIP, a set of 50.5K popular Java and Python GitHub repositories (v1) that contain all the metadata found in D_Pop, along with the source code. Java and Python are among the most common languages in GitHub. In the future, we intend to add more repositories in this level, that will be either language-agnostic datasets or focusing on different languages.

2.2.2 Repository Embedding and Retrieval.

The primary goal of using embeddings to represent repositories is to offer a more detailed and context-sensitive representation that surpasses keyword-based matching. With the use of repository metadata and source code, we can encapsulate the wealth of information present in repositories, thereby facilitating more effective search and retrieval operations. In our approach, we generate embeddings for all the repositories in our datasets. Previous studies have utilized NLP-based techniques [86, 84] to represent a GitHub repository within an embedding space. To construct the embedding for a repository, we consider all the available features of that repository in the corresponding dataset. Specifically, we have two types of features across our three datasets: (i) Metadata and (ii) Source Code. For each type, we first pre-process the data to eliminate unnecessary text, and then input them into the model to produce the embedding. Finally, we index these embeddings to support the retrieval process.

Feature Preprocessing. We consider different approaches to prepare the different meta-data features and the source code for the next step. We briefly describe them below.

- *Name.* Repository names often describe the purpose of the repository and usually consist of several words that are “combined” into one string. Name and string tokenization is a non-trivial task, which can be computationally intensive [89, 69]. We use delimiters (such as “-” and “.”) and Camel Case conventions to extract the tokens as followed in a recent study [69]. The final output is a string where we concatenate the tokens separated by spaces. For example, “telegram-bot-tutorial” becomes “telegram bot tutorial”.

- *Description.* The description field provides a concise understanding of the repository, and is usually 1/2 sentences. Some descriptions include potentially confusing information, such as URLs and code snippets [84]. We identify and remove such information to improve the relevance of the embedding.

- *Topics.* Topics are user-specified keyword-phrases with one or more words. The output is a string, which we obtain by concatenating the topics in the order they are provided.

- *Readme.* We extract first 300 characters, as most functionality is discussed in Readme as reported in recent studies [96]. Then, we remove any URLs and code snippets, as we did in Description.

- *Source Code.* We extract and combine source code from all files of a repository into one file, used to generate an embedding via CodeBERT [22].

Embedding Generation. We describe the creation of the embedding representations of the repositories using the available features at each level. We employ two distinct embedding approaches to capture the functionality of the repositories. The first approach focuses on repository metadata, while the second approach generates an embedding of the source code.

- *Text Embedding.* To capture the functionality of a GitHub repository from its metadata content, we use SentenceBERT [17], to represent repositories in the embedding space as used in recent studies [82, 84]. For each repository, we create a single document, *RepoInfo*, by concatenating all available textual metadata features (e.g., name, description etc), which is commonly done in such settings [86, 82, 59]. Finally, we employ SentenceBERT (all-MiniLM-L6-v2) to generate the embedding representations using the *RepoInfo*.

- *Code Embedding.* We utilize the CodeBERT approach [22] as followed in existing literatures [82, 59] to capture the information encoded in the source code of the repositories. We pass the preprocessed code through the CodeBERT model to generate embedding vectors for source code representation.

Indexing. We leverage the FAISS library (Facebook AI Similarity Search) to efficiently store and retrieve the closest vectors in the embedding space [20]. It is an optimized and efficient similarity search library to measure the similarity between embedding vectors. We create three indexing databases $D_{All_{faiss}}$, $D_{Popular_{faiss}}$, $D_{VIP_{faiss}}$, each corresponding to our datasets D_All, D_Pop, D_VIP, respectively, to store the embeddings of their repositories.

Retrieval. Given a query repository with its text and code embedding vectors, the retrieval task is to find a list of top-k similar repositories for each of our datasets D_All, D_Pop, and D_VIP, respectively. To retrieve the similar repository set, we compute the normalized

cosine similarity between the query embedding and the embeddings stored in the indexing databases, and retrieve the list of top-k similar repositories in order of cosine similarity. For the D_VIP dataset, we consider the average of text and code embedding similarity to make the final ordering of the result set. We can use concatenation/maximum/minimum of embeddings or other statistical metrics. We find that using average is widely preferred choice [104]. As D_All and D_Pop have only text embedding, we use it solely for the final scoring and ordering for them.

Re-Ranking. We include an optional re-ranking step to improve result relevance. Using LLM2Vec [6], an LLM-based dense embedding method built on Mistral-7B-Instruct [54], we embed the query and each retrieved repository using their *RepoInfo*, and re-rank results by cosine similarity.

The rationale for not using the LLM2Vec embedding from the beginning, instead of the S-BERT (all-MiniLM-L6-v2) embeddings, is motivated by the related cost. Using LLMs for all our ~ 300 million repositories would incur a prohibitively high computational [65] or monetary cost. In future, we will explore cost-effective LLM-based methods for this task.

2.2.3 Result Interpretation and Context.

The need for result interpretation arises from the challenges faced during repository searches. Obtaining a diverse set of repositories with varied purposes and functionalities, it is difficult to understand the scope and relevance of the search results. To address this, our methodology provides context about the search results in the form of: (a) clustering and (b) cluster summarization.

Gemma3-1B-it Prompt:

`<start_of_turn>user`

Given a cluster that contains the following documents:

$\{RepoInfo\ List\}$

The cluster is described by the following keywords:

$\{KEYWORDS\}$.

Based on the information about the cluster above, please create a short meaningful topic of this cluster.

`<end_of_turn>`

`<start_of_turn>model`

Cluster summary:

Gemma3-1B-it output: *summary*

Figure 2.3: Cluster summarization prompt for Gemma3-1B-it.

Hierarchical Clustering. We utilize dendrograms to capture the hierarchical structure of the retrieved repositories without assuming any a priori knowledge. Algorithmically, we use agglomerative hierarchical clustering following best practices [75], applied directly to the embeddings of the retrieved repositories.

- *Interactive Clustering.* The clustering approaches can be used in an interactive or alternating way to enable the user to discover clusters of interest at the granularity that makes sense for their investigation. For example, a user can use hierarchical clustering to identify the right level of clustering and then focus on specific clusters to analyze further.

Summarization. We provide a summarization method to help users understand the essence of a cluster. We employ an LLM to summarize a cluster given the *RepoInfo* of all the repositories in a cluster. We use the prompt template shown in Figure 2.3. To generate representative summarization, first, we extract a set of most frequent keywords from the *RepoInfo* of each repository [40]. Second, we then find the most frequent keywords across the whole cluster, and we select the repositories in order of decreasing frequency of the keywords. Though we can use any LLM of our choice, we use Gemma-3-1b-it [98], which is lightweight and efficient for inference. This is the final step and indicative results are shown in Figures 2.1 and 2.6.

2.3 Experiments and Evaluation

Given a query repository, we retrieve the top-k, denoted as TopK, similar GitHub repositories using RepoScope and our baselines.

Baselines. We use the GitHub REST API as the baseline to evaluate RepoScope. GitHub offers two keyword-search APIs: REST API [2] and GraphQL API [30]; we select GitHub REST API for comparison.

Research Questions. We answer the following questions.

RQ1: How does RepoScope compare to the GitHub API?

RQ2: How does RepoScope’s performance vary across the three levels of granularity?

RQ3: How does the size of the result set TopK affect RepoScope’s performance compare to GitHub REST API?

RQ4: How well does the clustering and summarization of RepoScope work?

Evaluation. Given the high cost and effort required for traditional human-based user studies, we leverage AI agents, specifically ChatGPT, as an alternative to validate our repository search results, as LLMs are found to produce inter-rater agreements close to human-raters in recent studies[1]. We also assess the accuracy of ChatGPT in this context on a set of sampled search results and their corresponding ChatGPT labels, as discussed in Section 2.3.7. Moreover, we employ a user study to understand the effectiveness and friendliness of RepoScope as a capability.

Evaluation Metrics. We introduce five metrics that are used in our evaluation. We use the first three metrics for RQ1, RQ2 and RQ3 and the two other metrics for RQ4.

Total True Positives is the total number of True Positives among all the results retrieved for all query repositories in QuerySet by any given method. We call a repository pair (query, result) as True positive when they are labeled as *similar* by ChatGPT as well.

Average Precision@k is defined as the proportion of True Positives among the top-k results.

Success Rate is the percentage of successful queries; a query is successful if any of the top-k results is True Positive.

Average Silhouette Score is used to quantify the consistency within and across the generated clusters [92].

BLANC score is a well-known reference-free metric for evaluating summaries [106].

Experimental Setup. To evaluate RepoScope, we construct QuerySet, a set of query repositories for our evaluation. Since GitHub lacks *example-based query*, we emulate it by selecting representative keywords for a given query repository and use them as keywords in

Table 2.2: Performance evaluation of RepoScope and the GitHub API for top-10 search results on our QuerySet.

Level	Method	Dataset	Avg. Precision@10 (%)	Total True Positives	Success Rate (%)
L1	GitHub API	All Repos	66	856	92
	RepoScope	D_All	86 ($\uparrow 20\%$)	1253	100
L2	GitHub API	Popular Repos	51	540	70
	RepoScope	D_Pop	75 ($\uparrow 24\%$)	1095	98
L3	GitHub API	Most Popular Repos	21	147	29
	RepoScope	D_VIP	59 ($\uparrow 38\%$)	867	95

Table 2.3: Performance evaluation of RepoScope and GitHub REST API in level L1 over different TopK values across randomly selected five queries from the QuerySet.

Level	k	Method	Dataset	Avg. Precision@k (%)	Total True Positives	Success Rate (%)
L1	20	GitHub API	All Repos	69	69	100
		RepoScope	D_All	88 ($\uparrow 19\%$)	88	100
	50	GitHub API	All Repos	66	164	100
		RepoScope	D_All	81 ($\uparrow 15\%$)	203	100
	100	GitHub API	All Repos	64	318	100
		RepoScope	D_All	73 ($\uparrow 9\%$)	367	100

GitHub REST API for retrieving similar repositories. Next, for each of the query repositories, we retrieve the top-k results from RepoScope per its datasets and the baseline, and use ChatGPT to validate the results.

Query Repository Set Selection. We selected *QuerySet*, a collection of 146 repositories, as our query-by-example repository input for the search process. Among them, 46 queries

were curated from [77] and the remaining 100 were manually selected following the criteria: (1) repositories must include the keywords “create” and “application” in their description, and (2) have a rating of at least 5 stars and 5 forks. These criteria were chosen to ensure that query repositories exhibit characteristics indicative of practical development scenarios.

From Query-by-Example to Keyword-Based Search. To compare RepoScope with GitHub REST API, we transform example-based queries into keyword-based queries that are compatible with the GitHub API. This is fair because it reflects the practical approach users must take in the absence of RepoScope. To identify representative keywords, we use three experienced computer scientists. One of them took the role of a *curator*, while the other two worked as *validators*. The curator creates an initial list of keywords for each of the QuerySet repositories. Then, the two validators check the initial set and suggest modifications: adding or removing keywords. Our study found that the validators mostly agreed with the initial keyword set. In fact, 93% of the time, at least one validator was in full agreement with the initial keyword set, while all three people agreed on the exact same keyword set 74% of the time. With this kind of validation, we were confident to use the initial keyword sets. *Note that using the union of two keyword sets will not necessarily give better similarity results.*

Generating and Filtering Results for Fair Comparison. We generated the top-k search results (excluding forks) for each query repository for all datasets in RepoScope and GitHub API. Because GitHub API offers filtering constraints, we generated two more search results to ensure a fair comparison with D_Pop and D_VIP. Specifically, we applied same filtering constraints to GitHub API that we used for creating D_Pop and D_VIP datasets (as

ChatGPT Prompt:

You will be presented with the information of two GitHub repositories and your job is to provide only one of the labels from the following list based on the functionality of the given repository pairs.

Please consider the high-level functionality of the repositories, regardless of the programming languages or technologies used. The high-level main functionality of the repositories must be matched to be similar, otherwise they will be neutral/dissimilar.

Choose ONLY from the list of labels provided here:

- (i) *similar*
- (ii) *neutral*
- (iii) *dissimilar*

Repository 1: {repo name}, {description}, {readme}

Repository 2: {repo name}, {description}, {readme}

ChatGPT output: *similar / neutral / dissimilar*

Figure 2.4: ChatGPT-4 prompt for labeling a pair of repositories (query, result) based on their functional similarity.

discussed in Section 2.2). We refer to the resulting sets as GitHub REST API with *Popular Repos* (Level L2) and *Most Popular Repos* (Level L3). The unfiltered baseline results are denoted as *GitHub All Repos* (Level L1).

ChatGPT-Based Functional Similarity Annotation. For each (query, result) repository pair, we used the ChatGPT API (gpt-4.1) to classify the result repository as functionally *similar*, *neutral*, or *dissimilar* to the query repository. The prompt used is shown in Figure 2.4. We follow the best practices for annotation tasks [28] and set the temperature to 0.2 with other suggested ChatGPT settings. Our process involved providing ChatGPT with relevant metadata about the repository, including its name, description, and readme content.

2.3.1 RQ1: RepoScope vs. the GitHub API.

As we report in Table 2.2, RepoScope outperforms GitHub API across all metrics when evaluated on the D_All and D_Pop datasets for TopK parameter $k = 10$. It surpasses GitHub API in 70% of the QuerySet repositories, ties on another 24%, while only in 6% of them does GitHub API show better results than RepoScope. With datasets D_All and D_Pop, RepoScope achieves average precision@10 of 86% and 75%, respectively, approximately 20% and 24% higher than the 66% and 51% achieved by GitHub API at level L1 and L2, respectively. Additionally, RepoScope returns 1253, 1095, and 867 True Positives across its datasets, compared to 856 for GitHub API with level L1. Moreover, GitHub API offers 92% success rate in level L1, whereas RepoScope has at least 95% success rate, outperforming GitHub API at all levels.

Observation 1. *RepoScope consistently outperforms GitHub API at all levels, achieving $\sim 20\%$ better precision at level L1 with 100% success rate.*

When comparing the GitHub API in Level L2 and L3 with RepoScope under the same constraints as our D_Pop and D_VIP datasets, GitHub API achieves 51% and 21% precision, respectively. RepoScope with D_Pop (75%) and D_VIP (59%) outperforms GitHub API by 24% and 38% respectively, in terms of precision. Additionally, RepoScope with D_VIP in level L3 achieves 867 True Positives which is 5.9 times as many as the GitHub API 's 147 True Positives with a 29% success rate, compared to of 95% of D_VIP.

2.3.2 RQ2: Efficacy at Different Levels of Granularity.

As detailed in Table 2.2, RepoScope with D_All achieves an average precision@10 of 86%, which is 11% and 27% better than that with D_Pop (75%) and D_VIP (59%), respectively for TopK parameter $k = 10$. Notably, RepoScope in Level 1 (dataset D_All) identified 14.4% and 44.5% more True Positives compared to the other datasets respectively. On the other hand, although D_All is ~ 200 times larger than D_Pop, D_Pop achieves a 98% success rate in its top-10 results. D_VIP lags behind D_All by only 30.8% in True Positives, despite being $\sim 5782\times$ smaller.

Observation 2. *RepoScope with dataset D_All achieves superior performance, while D_Pop and D_VIP demonstrates strong efficiency despite their significantly smaller sizes.*

2.3.3 RQ3: Efficacy at Different Result Size TopK.

To assess performance metrics for varying sizes of result sets, we uniformly selected five queries at random from the QuerySet. We then retrieved results for different values of TopK, where $k = 20, 50$, and 100 , and collected performance metrics for level L1 using GitHub API and RepoScope. Table 2.3 demonstrates that RepoScope consistently surpasses GitHub API in every metric for all TopK values. For $k = 20, 50$, and 100 , RepoScope achieves 88%, 81%, and 73% Average precision, respectively, while GitHub API achieves at most 69% across any k value. In addition, RepoScope provides at least $\sim 15\%$ more True Positives than GitHub API.

Observation 3. *RepoScope outperforms GitHub API in level L1 by at least 9% better precision in different TopK results.*

2.3.4 RQ4: Result Clustering and Summarization.

We assess the clustering and summarization of RepoScope. We use the same set of five randomly selected queries from QuerySet that we used for evaluating RepoScope’s efficacy at different TopK values. We create clusters of search results for each of these five query repositories based on their repository embeddings. We measure the quality of clustering and summarization using the average Silhouette score and BLANC metrics. RepoScope achieves an average Silhouette score of 0.54 for the generated clusters, which is considered as a good clustering in the literature [16]. RepoScope also achieves a BLANC score of 0.58 for the summaries, within the 0.5–0.7 range typical of high-quality summaries [106].

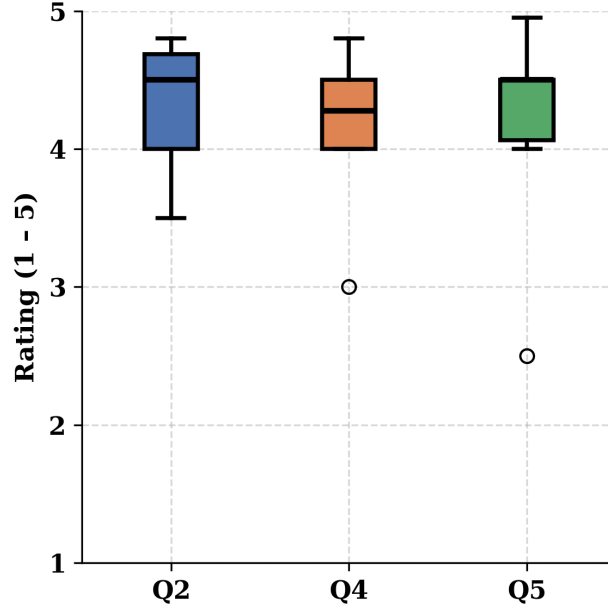


Figure 2.5: User ratings on using RepoScope for Q2, Q4 and Q5.

Observation 4. *RepoScope effectively clusters and summarizes similar repositories. It achieves strong Silhouette score (0.54) and BLANC (0.58) scores, demonstrating its ability to capture meaningful relationships in repository search.*

2.3.5 User Study to Evaluate RepoScope.

We conduct a small-scale, task-based user study to gather early, practitioner-oriented feedback on our method RepoScope, for finding and exploring similar GitHub repositories. We recruit ten participants (1 Masters, 8 PhD students and 1 Industry Professional), where each participant has at least five years of active GitHub experience for searching of code at least once per day. The procedure consists of a brief tutorial followed by searching similar repositories, for two query repositories; *nehamchangappa/Online-Banking-Web-Application-*

Project[32], that implements a banking web application, and *davidson679/Bypass-Web-Application-Firewalls* [33], which is a penetration testing project. For each scenario, participants perform the following tasks: (a) submits a query repository, (b) inspects the ranked list returned by RepoScope, (c) examines the interactive cluster view. After using RepoScope, participants complete a short questionnaire consisting of five items to provide rating. *Notably, they are asked to play with other query repository of their interest as well.*

Q1: RepoScope is easy to use (Agree/Disagree).

Q2: Top-10 results returned from RepoScope are similar to the query repository (1-5).

Q3: Clusters and their summaries are useful and meaningful (Agree/Disagree).

Q4: Clusters from RepoScope are internally coherent and clearly separated (1-5).

Q5: I am overall satisfied with the RepoScope (1 – 5).

User Feedback. Our evaluation reveals that RepoScope is found to be easy to use by all participants. We plot their ratings across Q2, Q4, and Q5 in Figure 2.5. Ratings for Q2 are tightly around rating (4-5). This suggests that the ranked search results are deemed relevant to the participants for the given query in most cases. The overall mean score for user satisfaction is 4.2 out of 5, with nine participants expressing satisfaction at or above 4, indicating a strongly positive overall impression. In addition, participants provide feedback on the clustering and summarization features. All participants agree that the visualizations help them explore the result space, and their average scores for cluster coherence and separation are consistently above 4.

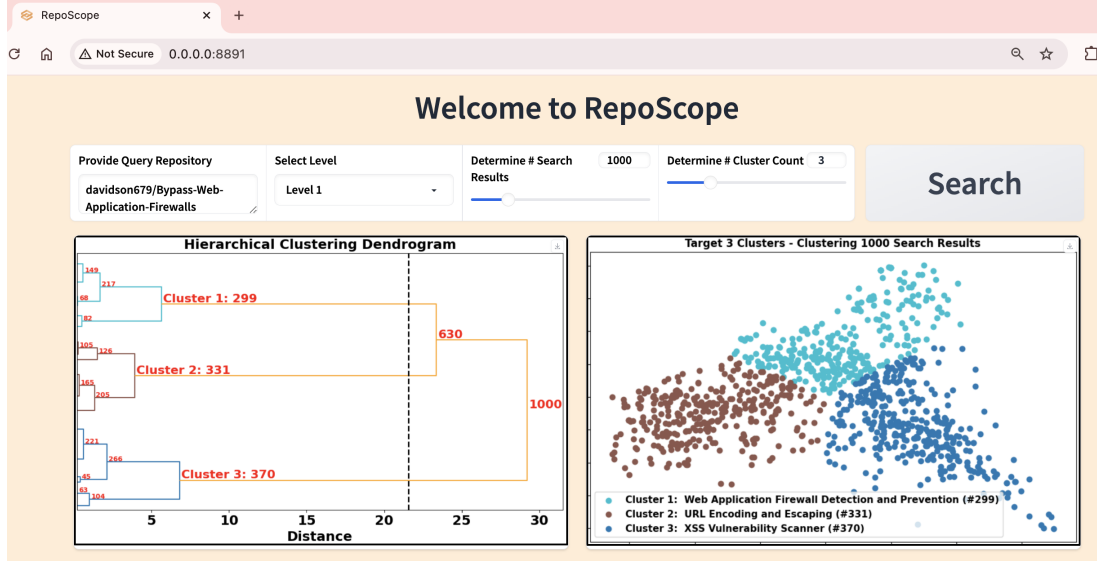


Figure 2.6: Contextual information: RepoScope returns 1K repositories for the query repository *davidson679/Bypass-Web-Application-Firewalls* and provides meaningful context: (a) clustering hierarchy of the returned repositories as dendrogram, and (b) cluster profiles: summaries and a two dimensional projection of the three clusters that the user selected (dotted vertical line on dendrogram).

2.3.6 Search Latency and Storage Requirement.

User Response Time of RepoScope. We provide a brief analysis on the performance of our platform as seen by a user. We calculate the average, median and maximum search time per query to retrieve the top-10 similar repositories for each of our datasets. An answer at level L1, using D_All, takes an average of 7.9, median of 7.8 and maximum 8.4 minutes per query. At level L2 using D.Pop, we get results within 0.6-0.7 minutes with a median of 0.66 min. At level L3 using D.VIP we get results within 0.55-0.56 minutes. Level L1 queries take longer because of the size of the database and the 384 dimensions of the embedding space. In addition, the re-ranking method with Mistral-7B takes 0.5 minutes per query for the top-10 results. We plan to reduce the response time to under a minute even for level L1 queries. We will consider well-known dimensionality reduction approaches [95] and

Table 2.4: Data collection and storage for the datasets in RepoScope.

	D_All	D_Pop	D_VIP
Data Collection (Days)	27	1.2	7
Raw Data Storage (GB)	60	9.6	77
Embedding Storage (GB)	90	1.7	0.17

lower-cost reranking LLM models [113], but in a way that will not degrade the quality of the results. However, we ran the experiments in a Linux machine with 64GB RAM, 16 CPU cores and 2 Tesla V100-16GB GPUs.

Data Collection and Storage Cost. We used GitHub REST API to collect metadata and source code for our datasets. It took 27, 1.2 and 7 days, for collecting them for D_All (60GB), D_Pop (9.6GB) and D_VIP (77GB), respectively. After collecting them, we created embeddings for each of the repositories with their associated metadata (and source code in D_VIP). Then, we created indexing for them that incurred storage of 90GB, 1.7GB and 0.17GB for the datasets, respectively. We show the details in Table 2.4.

2.3.7 Evaluating ChatGPT in Labeling Similar Repositories.

We evaluate ChatGPT’s accuracy and consistency in labeling similar repository pairs. Three experienced computer scientists independently reviewed 50 randomly selected annotations from each category, with final labels determined by majority vote. The validators agreed with ChatGPT labels 82% of the time for similar pairs and 90% for dissimilar pairs, resulting in an overall accuracy of 86%. We measured interrater agreement using a Kappa Coefficient of 0.866, which indicated a strong agreement [108].

On the other hand, we also assessed the consistency of ChatGPT in labeling the same pair multiple times. To explore this, we considered 100 different pairs of (query, result), repeating each pair five times. We selected each of 100 repositories from QuerySet as a query and randomly selected a result repository from the RepoScope results. The labels generated from ChatGPT were highly consistent: 94% of the pairs received the same response every time, while only 5% had one response that differed among the five. These findings suggest that ChatGPT demonstrates a high degree of reliability.

2.4 Discussion

Using RepoScope: a Publicly Accessible Platform. We will release RepoScope as a publicly accessible platform as shown in Figure 2.6. Due to the computational demands of search time and storage, we will offer RepoScope in two distinct modes; (a) direct API endpoints: users can avail searching across Level 2 and Level 3, (b) tool and datasets: users can download the RepoScope tool and datasets, allowing for comprehensive offline exploration of the GitHub search space. This will enable users to start with a repository of interest and conduct either focused similarity searches or broad exploratory searches. In focused search, users may seek the top-30 most similar repositories; in exploratory search, they may instead map the surrounding landscape and identify repository “families” based on intent, functionality, deployment target, or other technical characteristics. These modes can also be combined interactively and iteratively. Furthermore, RepoScope provides hierarchical clustering and summarization, facilitating a deeper understanding of the search results.

The Usefulness of the Three Granularities. It should be clear that the three levels of RepoScope have their own interest and they could be appropriate for different types of investigations. The first level could be useful for studies that want to explore the landscape of activity across the whole of GitHub. Naturally, a study at this level trades off accuracy for covering all the repositories, which is very difficult to do directly through the GitHub REST API. The second level provides more insights for influential repositories. Finally, the third level trades off coverage but is suitable for a study that would want to consider the code of the repository.

How Frequently Should We Re-crawl GitHub? This is an essential question for ensuring that RepoScope will always have relatively fresh and relevant information. First, we want to mention that collecting snapshots at any interval could be useful for a longitudinal study. Second, we conducted two distinct data collections, one year apart as shown in Table 2.1. We found that the total number of repositories increased by 22%, while the increase is lower at the other two levels (10% and 5%). Trying to balance cost and utility, we believe that collecting the data once every four to six months could provide sufficiently close monitoring of the growth.

Incorrect Results. Despite RepoScope’s effectiveness, it occasionally misidentifies repositories as similar for certain queries. In certain cases, it failed to retrieve similar functionality repositories due to confusions with repository names or other semantic ambiguities. For example, the query repository *dragome/dragome-sdk* - a SDK tool for creating client-side web applications - is mistakenly confused with the feature “*drag-and-drop*” in the description of the retrieved repository *thirdweb-example/makeswift*, a web3 application utilizing a drag-

and-drop web builder. Another query example that yields incorrect results is *livrospring-mvc/lojacasadocodigo*, a sample e-commerce application built using Spring MVC, which is found to be similar to *springhispano/website*, a web application for content management and community engagement. Although both repositories contain the keyword “MVC”, they differ in language, with the former written in Portuguese and the latter in Spanish. To address the storage and processing costs, we had to opt for the lightweight S-BERT embedding - “all-MiniLM-L6-v2” model (dimension=384) with limited performance compared to models specifically designed for advanced multilingual applications. We propose exploring more sophisticated embedding and dimensionality reduction techniques to address these issues in our future work.

2.5 Threats to Validity

Ensuring the Capability of LLMs. We use LLMs (ChatGPT specifically) to evaluate our study and revisit our effort to ensure that ChatGPT provides accurate results. Recall that we use a manual evaluation process to validate that ChatGPT identifies similar repositories with an average accuracy of 86%. This level of accuracy is sufficient for our study, and ChatGPT is a reasonable way to validate our approach [1].

The Representativeness of Our Query Set. Ensuring the use of a representative dataset is always a challenge for any empirical study. In our evaluation, we used a set of 146 queries. We required popular repositories which is a commonly-used practice that attempts to ensure that the repositories are not trivial, are created intentionally and with some attention and thought [38, 57]. Upon investigation, we also find that our repositories have

a reasonably wide range of topics and technical domains. First, they cover several topics and domains including: web and mobile applications, network security and malware tools, machine learning, vision and visualization tools, DevOps, trading bots, cloud applications, and computer hardware. Second, they cover a good range of platforms including Windows, macOS, and Linux, providing a reasonable range of distinct repository types.

2.6 Related Work

No prior searching mechanism addresses the problem as we frame it. We group prior work into two categories: platforms and datasets, and similarity methods.

2.6.1 Platforms and Datasets.

Apart from GitHub, there are several noteworthy studies and platforms that collect and archive repository information. However, they have one or more of the following limitations: (a) they do not offer query-by-example repository search, (b) they rely on keyword-based searches, (c) they do not support multi-granularity search, or (d) do not provide context about the search results. While GHTorrent [39], GHArchive [24] and WoC[66] seem to focus exclusively on GitHub event data, such as commits, but they do not seem to support similarity searches, let alone query-by-example searches. On the other hand, platforms like GHS [15] and Software Heritage [18] support only keyword-based queries that could be less powerful than the GitHub API. None of these platforms support GitHub-wide multi-granularity query-by-example search or provide structured insights from the search results.

Table 2.5: Comparison of RepoScope with other related approaches.

Approach	Coverage	Example Query	Semantic Search	Multi-Granular	Cluster
GitHub	420M+	x	x	✓	x
GHS [15]	1.7M+	x	x	x	x
WoC [66]	209M+	x	x	x	x
Software Heritage [18]	375M+	x	x	x	x
RepoScope	292M+	✓	✓	✓	✓

Overall, previous efforts do not address the challenges that we do in our work. We illustrate them in Table 2.5.

2.6.2 Establishing Similarity Between Software Repositories.

Many studies focus on the algorithmic aspect of establishing the similarity between repositories, but they are not designed to support large scale exploration of software repositories. We group previous work according to their main algorithmic approach. First, several efforts use topic modeling on metadata and source code [56], [103]. Second, some methods use association rules [101], [102] to categorize software repositories. Third, there are several efforts that use embedded representations: Repo2Vec [86], Paper2Repo [93], Rep2Vec [82], and Topical [59]. Fourth, some efforts, such as CrossSim [77, 78], and Rep2Vec [82] introduce a graph-based approach to compute repository similarity by embedding source code and social interactions (such as an author starring a repository).

2.7 Conclusions

We introduce RepoScope, a query-by-example framework for GitHub repository search. RepoScope provides a comprehensive and scalable search at three levels of granularity designed to balance search scope and comparison detail. Specifically, for given a repository of interest, RepoScope will: a) search for functionally similar repositories through an efficient embedding-based approach, b) operate at three pre-downloaded datasets with different levels of granularity, and c) provide context and hierarchical clustering of the retrieved repositories. Our evaluation shows that RepoScope outperforms the baseline in 70% of the queries and achieves at least the same performance in 94% of cases. Besides, RepoScope provides 86% average precision within top-10 results with 100% success rate, whereas baselines achieve maximum 66% precision.

In the future, we plan to continue our work and establish it as a continually updated online capability. From a performance point of view, we will explore different algorithmic choices (such as embedding techniques) to reduce the response time especially for D_All repositories. Besides, we would like to explore the impact of metadata evolution and new repository additions on search performance.

We intend to open-source our code and datasets to maximize the impact and facilitate follow up research.

Chapter 3

MAGNET: A

Software-Engineering View of the Malware Ecosystem on GitHub

How does the malicious hacker community build malware repositories on GitHub? Prior work shows that GitHub hosts thousands of public repositories containing malware [85, 97], but less is known about what happens after these repositories appear on the platform. This gap matters because GitHub, with over 180 million developers and 630 million repositories as of 2025 [35], provides collaboration, distribution, and visibility mechanisms that can support both legitimate software development and malicious activity. Thus, the central question is not only whether malware repositories exist on GitHub, but what software-engineering and social patterns characterize their presence on the platform.

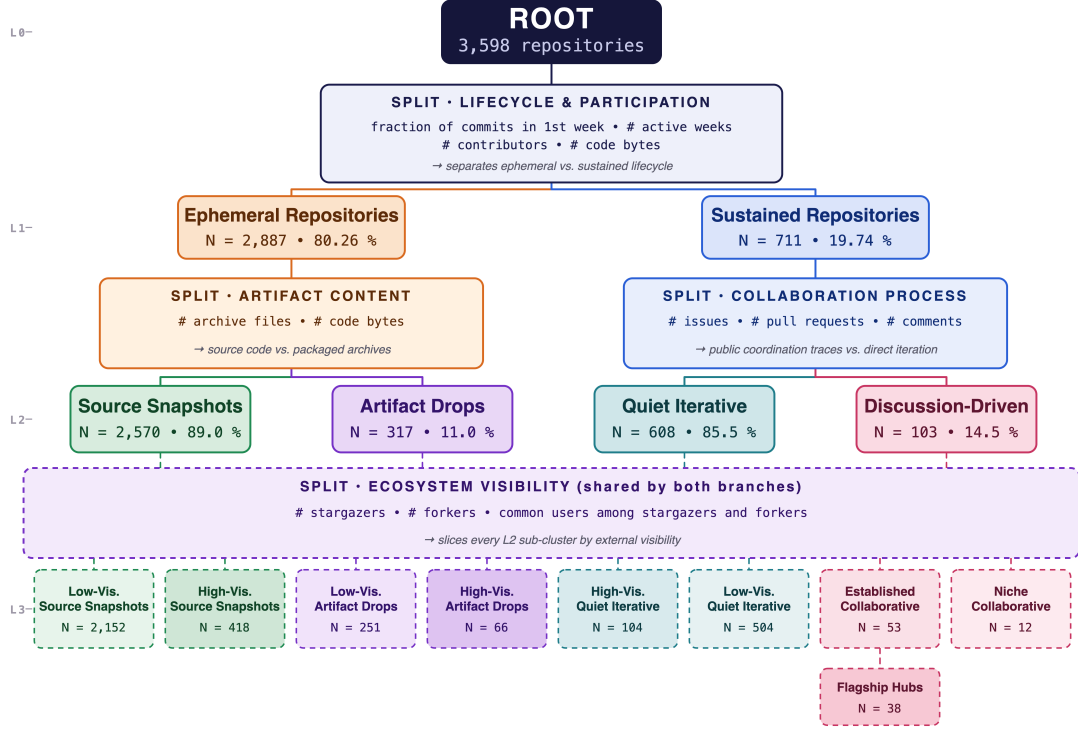


Figure 3.1: Decomposition of the 3,598 malicious-intent repositories into nine engineering archetypes via three orthogonal splits. **L1** separates *Ephemeral* ($N=2,887$, 80.3%) from *Sustained* ($N=711$, 19.7%) repositories on lifecycle and participation features. **L2**: artifact content splits Ephemeral into *Source Snapshots* and *Artifact Drops*; collaboration process splits Sustained into *Quiet Iterative* and *Discussion-Driven*. **L3** applies the ecosystem-visibility features to all four L2 clusters.

Background. Recent campaigns highlight the operational role of GitHub in this ecosystem.

In late 2024, Microsoft Threat Intelligence documented a large-scale campaign in which attackers created GitHub repositories to host first-stage info-stealer payloads, compromising close to one million devices worldwide [71]. Concurrently, the *Stargazer Ghost* campaign fabricated more than 3,000 GitHub accounts to manufacture stars and forks, using the platform’s reputation signals to make malware repositories appear legitimate [99]. In both cases, attackers relied not only on GitHub’s hosting capacity, but also on its software-development and reputation mechanisms. These cases show that effective monitoring must consider both malicious code and the repository-level behavior surrounding it.

Problem. How does the malicious hacker community build malware repositories on GitHub? This is the question that motivates this work. Answering this question requires separating malicious use from benign security work, since the same malware behavior may appear in research, education, defense, or abuse. We use the term *malicious-intent repository* for a public GitHub repository whose code implements malware behavior (such as credential theft, ransomware, DDoS, persistence, or command-and-control) and whose framing and authorship show no clear research, educational, or defensive intent, so the repository can support cyber-crime. We define this more precisely in Section 3.1. The input to our problem is the set of GitHub-level signals around these repositories: code, metadata, commits, issues, pull requests, comments, stars, forks, and the author’s wider portfolio. The desired output is a software-engineering view of the malicious side of GitHub.

Related Work. There has been limited work on the problem as we formulate it here. Prior work has mainly approached the malware repository ecosystem from three directions. First, detection studies identify which GitHub repositories contain malware [85, 82, 100]. Second, actor-profiling studies examine who creates these repositories and how those actors connect across forums and platforms [49, 97, 91, 80]. Third, code-centric studies analyze malware source code itself, including complexity, language distribution, and reuse [9, 10]. These efforts are important, but they do not provide a systematic software-engineering characterization of malicious-intent repositories on GitHub. In particular, we lack a unified account of their development, maintenance, and user-facing activity. We discuss related work in more detail in Section 3.6.

Contribution. We propose MAGNET, a systematic framework for studying malicious-intent malware repositories as a software ecosystem. The key novelty of MAGNET is the integration of three complementary stages. First, MAGNET identifies malicious-intent repositories using an LLM-agent pipeline that reads both repository and author-portfolio evidence. Second, it organizes the corpus into engineering archetypes using a hierarchical decomposition of development and maintenance behavior. Third, it models user engagement at three levels (*roles*): (a) co-developer, (b) supporter, and (c) follower, distinguishing a user who commits code from one who reports an issue or stars a malware repository. Collapsing these events into a single graph would hide this structure.

Research questions. We organize the chapter around three research questions.

- **RQ1: Identification.** How can we identify malicious-intent malware repositories?
- **RQ2: Development modes.** What engineering modes characterize malicious-intent repositories?
- **RQ3: Ecosystem behavioral patterns.** How is user participation organized across co-developer, supporter, and follower?

Key results. Our key results can be summarized in the following points.

- a. **MAGNET identifies 3,598 malicious-intent repositories.** Starting from 16,726 candidate repositories, the LLM-agent pipeline combines repository evidence with author-portfolio evidence. In a blinded manual validation, two validators jointly confirmed 97 of 100 sampled labels.

b. Growth concentrates after 2016, and ransomware becomes visible. Across the 2008–2022 timeline, 81.4% of the 3,598 malicious-intent repositories were created from 2017 onward. KEYLOGGER and BACKDOOR remain the most frequent types, while RANSOMWARE grows from 14 repositories in 2014–2016 to 185 in 2020–2022.

c. 80.24% of repositories undergo mostly ephemeral development. 2,887 repositories (80.24%) are one-shot or short-lived uploads, while 711 repositories (19.76%) are sustained. Within the sustained slice, 103 Discussion-Driven repositories concentrate 24.9% of all issues and 72.9% of all pull requests from the corpus.

d. Following communities are much larger than co-development communities.

The user activity around sustained repositories is concentrated in the follower role: 48,418 users appear through forks and stars, compared with only 1,100 users in co-developer. This gap shows that the ecosystem is much more visible as an audience surface than as a co-development space.

Our work in perspective. The proposed work is a step toward understanding the malicious side of public software-hosting platforms through a software-engineering lens. We intend to release MAGNET, the curated corpus, the per-repository evidence, and the labelling pipeline to support future research. Our goal is to provide a foundation for monitoring, threat intelligence, platform governance, and follow-on studies of malicious software ecosystems beyond GitHub.

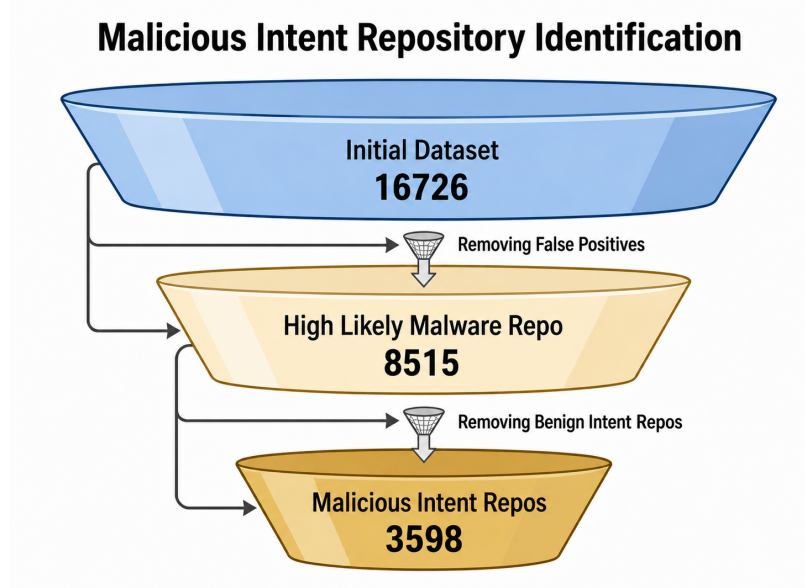


Figure 3.2: Malicious-intent corpus identification. Starting from 16,726 candidates, two evidence-based filters yield 3,598 high-confidence malicious-intent repositories.

3.1 Identifying Malicious-Intent Repositories

We construct a high-confidence corpus of malicious-intent GitHub repositories. Prior corpora [97, 85] conflate three groups: anti-malware tools that merely use malware terminology, code framed as defensive, research, or educational, and malicious-intent repositories whose code can support cyber-crime with no benign framing. We focus on the last group.

Figure 3.2 shows our pipeline. Starting from 16,726 candidate repositories, we apply two filters: the first removes false positives (repositories that do not implement malware behavior), and the second removes benign-intent repositories (those framed as research, education, or defense). The result is MALINTENT, a corpus of 3,598 malicious-intent repositories. Both filters are driven by structured judgements produced by an LLM agent.

Role. You are a malware-repository analysis agent.

Evidence provided. For each target repository, you receive (i) repository metadata, README, file tree, and selected file contents, and (ii) the owner’s profile together with a summary of the owner’s other repositories.

Judgements to produce.

- whether the code implements malware behavior;
- which malware type or types are present, if any;
- whether the author profile carries explicit research/educational/defensive sign;
- the inferred intent of the target repository; and
- the inferred intent of the owner’s wider portfolio.

Output format. Return one JSON object containing the five judgements.

Figure 3.3: Prompt template used by the LLM agent

3.1.1 Definition

We use the term *malicious-intent repository* for a public GitHub repository whose code, framing, and authorship together point to malicious use. Such repositories can support cyber-crime. The definition has three parts: (a) the code implements at least one malware behavior, such as ransomware, DDoS, or phishing; (b) the framing shows no clear research, educational, or defensive purpose; and (c) the author’s wider portfolio is not benign.

3.1.2 Candidate corpus and evidence

We start from 16,726 malware-candidate repositories obtained from prior author analysis work [97]. We cap at 2022 to give each repository time to accrue development and engagement traces. For each repository, we collect the GitHub-level evidence needed to reason

about both the code and its surrounding context: README, file inventory, commits, issues, pull requests, comments, stargazers, forks, contributors, the owner’s profile, and a summary of the owner’s other repositories. The author-level context matters because a single repository can be ambiguous when read in isolation.

3.1.3 LLM-agent labelling

For each candidate repository, we fetch an evidence bundle from GitHub including metadata, the README ($\leq 24\text{K}$ chars), the file tree (up to 400 paths), up to 15 source files (priority entry points, text extensions only), and the owner’s profile plus metadata cards for up to 30 of the owner’s other repositories — and pass it to an LLM agent (`gpt-5-mini-2025-08-07`, `temperature=0.2`). Repositories inaccessible at fetch time are excluded from MALINTENT. The agent runs under the prompt template in Figure 3.3; we combine the five judgements into the two-stage filter shown in Figure 3.2.

Stage 1: Removing false positives. We discard any repository whose code does not implement malware behavior. This removes anti-malware tools, signature lists, and other content that uses malware vocabulary without implementing any malicious functionality. Of the 16,726 candidates, 8,515 pass this stage.

Stage 2: Removing benign-intent repositories. Among these, we keep only those framed as malicious. A repository passes if (i) the owner’s profile shows no explicit research, educational, or defensive signal, and (ii) either the repository is labelled malicious and the author is not benign, or the repository is labelled likely-malicious and the author is also malicious. This stage yields MALINTENT, a corpus of **3,598 high-confidence malicious-intent repositories from 3,487 authors**.

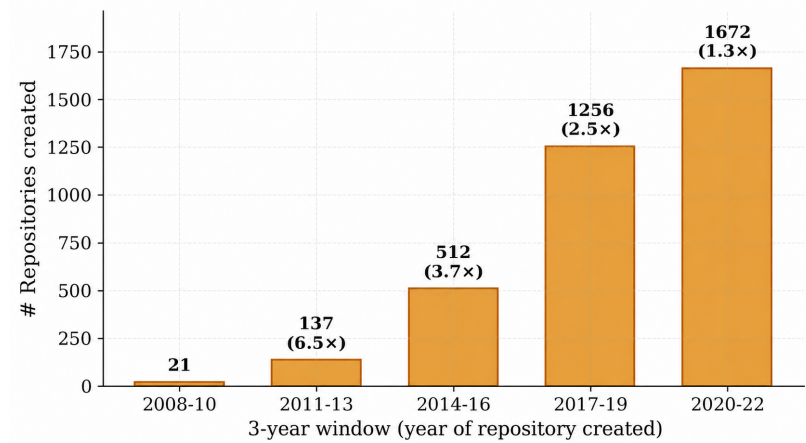


Figure 3.4: Repository creation by three-year window. The corpus grows in multiplicative waves and is concentrated after 2016.

3.1.4 Manual validation

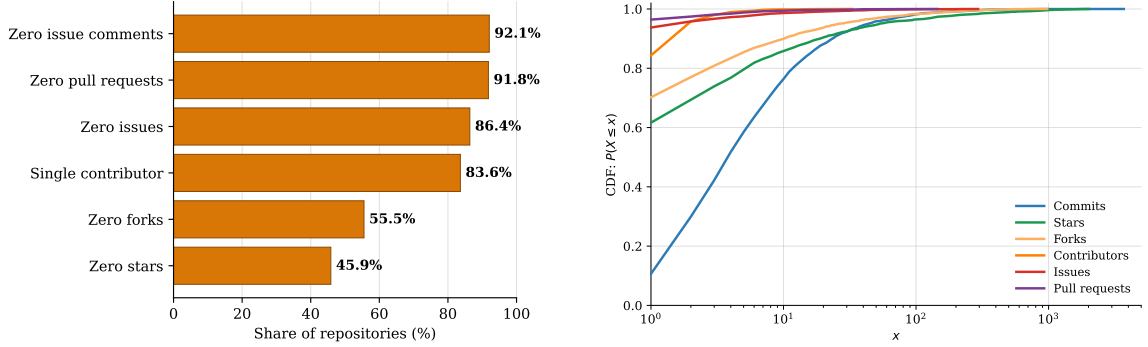
Two researchers independently re-labelled a blinded random sample of 100 repositories from MALINTENT using the same evidence bundle and §3.1.1 definition. Annotators A and B confirmed 99/100 and 98/100 labels, respectively, with 97/100 joint confirmations, yielding 97% precision on the LLM-positive sample (95% Wilson CI: [91.5%, 99.0%]) and high agreement (Gwet’s AC1 $\approx$ 0.97 [41]).

Observation 5. *From 16,726 candidate repositories, MAGNET identifies 3,598 malicious-intent repositories, with 97% human-validated precision.*

3.2 Trends and Growth

Using MALINTENT, we examine how the malicious-intent ecosystem evolves on GitHub:

- (i) overall growth and (ii) shifts in malware types.



(a) Share of repositories with zero events on each engagement signal (“Single contributor” counts exactly one).

(b) Cumulative distribution functions for six engagement metrics. For each metric, the curve plots $P(X \leq x)$, the fraction of repositories with at most x events of that type.

Figure 3.5: Engagement distribution across the 3,598 malicious-intent repositories. (a) Most repositories are silent on engagement signals: 91.8% have zero pull requests, 86.4% have zero issues, and 83.6% have a single contributor. (b) Commits rises slowly from 0.10 at $x = 1$, indicating the widest spread of values, while issues and pull requests saturate near 1.0 almost immediately, indicating that the great majority of repositories accumulate little or no public coordination.

3.2.1 The growth of the malware ecosystem

Figure 3.4 groups repositories by publication year into three-year windows. The corpus grows in multiplicative waves: from 21 repositories in 2008–2010 to 137 in 2011–2013 ($6.5\times$), 512 in 2014–2016 ($3.7\times$), 1,256 in 2017–2019 ($2.5\times$), and 1,672 in 2020–2022 ($1.3\times$). The growth factor moderates in the most recent window, but the absolute volume stays historically high. 81.4% of MALINTENT was published from 2017 onward, making malicious-intent development on GitHub a recent and rapidly expanding phenomenon.

Observation 6. *The ecosystem expanded sharply and remains historically large, with 81.4% of repositories created after 2016, although the multiplicative growth rate moderates in the most recent window.*

3.2.2 The growth by malware type

The composition is skewed toward a small set of malware types. KEYLOGGER (891 repositories, 24.8%) and BACKDOOR (659, 18.3%) lead consistently across the observation period; together they cover 1,550 repositories, which is 43% of MALINTENT. BOTNET, TROJAN, RANSOMWARE, and DDOS follow at 7–9% each. About 4.4% of repositories are multi-labeled (e.g., BOTNET+DDOS, BACKDOOR+KEYLOGGER).

RANSOMWARE is the emerging malware type. It is absent before 2014, appears with 14 repositories in 2014–2016, grows to 79 in 2017–2019, and reaches 185 in 2020–2022, becoming one of the top types in the final window.

***Observation 7.** The most frequent types are KEYLOGGER and BACKDOOR, accounting for 43% of MALINTENT; RANSOMWARE is the emerging type.*

3.3 Engineering Archetypes

How are these 3,598 malicious-intent repositories actually developed and used on GitHub?

We answer this question by organizing the corpus into engineering archetypes via a hierarchical decomposition. Each level asks one orthogonal question over a focused set of GitHub metadata features, yielding the nine-archetype taxonomy of Figure 3.1. The corpus is sparse on most engagement signals: 91.8% of repositories have zero pull requests, 86.4% have zero issues, and 83.6% have a single contributor (Figure 3.5). The remaining minority carries nearly all observable engineering activity, and within that minority the operating modes diverge sharply. The taxonomy reflects this layered structure: it first separates the active minority from the silent majority, and then identifies modes within each branch.

Table 3.1: Features used at each level of the taxonomy, grouped by conceptual axis. All features are computed from GitHub metadata after bot filtering.

Axis	Features
Lifecycle	first-week commit share; no. of active weeks; no. of contributors
Content	no. of code bytes; no. of archive files
Coordination	no. of issues; no. of PRs; no. of comments
Visibility	no. of stargazers; no. of forkers; no. of overlapping forkers-stargazers/repo

3.3.1 Features and clustering procedure

Each level of the taxonomy uses a focused feature set drawn from one or more of four conceptual axes: *lifecycle* (when commits land and who makes them), *content* (what is shipped), *coordination* (how the team interacts publicly), and *visibility* (who pays attention). Table 3.1 lists the features per axis; all are computed from GitHub metadata after bot accounts are filtered out.

At each split we log-transform the relevant features to stabilize the heavy tails typical of GitHub activity metrics, robust-scale them by median and interquartile range to limit outlier influence, and apply Ward agglomerative clustering [111] on the resulting Euclidean distance matrix. We use $k=2$ at every split except the L3 refinement of the Discussion-Driven cluster, where $k=3$ yields three archetypes. Silhouette scores [87] are 0.589 at L1, 0.756 at L2-Ephemeral, and 0.755 at L2-Sustained. A 1,000-permutation null yields maximum silhouettes of 0.13, 0.11, and 0.21 respectively ($p<0.001$).

Table 3.2: Level-2 archetypes and signatures.

Archetype	Repos	Share	Signature
Source Snapshots	2,570	71.4%	Short source-code burst; median < 1 day commit span.
Artifact Drops	317	8.8%	Archive/binary-heavy; near-zero median source-code bytes (median < 10 min commit activity).
Quiet Iterative	608	16.9%	Persistent activity; limited public discussion.
Discussion-Driven	103	2.9%	Issues, comments, PRs, external users join here.

3.3.2 The three-level taxonomy

We show our three level taxonomy in Figure 3.1. The L1 split asks whether a repository is sustained over time. **Ephemerally developed repositories** (2,887, 80.24%) have a median first-week commit fraction of 1.00, a median of one active week, and a median of one contributor. **Sustained repositories** (711, 19.76%) distribute their commits across a median of four active weeks and have two contributors.

The L2 split separates the operating modes inside each branch. Among Ephemeral repositories, **Source Snapshots** (N=2,570) contain source code with a short lifecycle, while **Artifact Drops** (N=317) ship packaged artifacts with near-zero source code. Table 3.2 summarizes the four Level-2 groups.

Among Sustained repositories, **Quiet Iterative** repositories (N=608) iterate through committing code with little public coordination (median twelve active weeks, ≈ 0 issue com-

ments), while **Discussion-Driven** repositories (N=103) accumulate substantial coordination traces (median fourteen issue comments; this cluster alone holds 24.9% of all issues and 72.9% of all pull requests in the corpus). The L3 split applies the same ecosystem-visibility features to each L2 cluster. Three L2 clusters split at $k=2$ into low- and high-visibility variants; Discussion-Driven splits at $k=3$ into *Niche Collaborative*, *Established Collaborative*, and *Flagship Hubs*, reflecting its more interactive character.

3.3.3 The four engineering modes

Source Snapshots are the default mode. This archetype is the default way malicious-intent code is published on GitHub: dropped once and left rather than developed or operated. The median repository has a lifespan of under a day, and 84.7% never receive an issue, pull request, or archive after creation. The median repository contains only ≈ 6 KB bytes of source code, about the size of a single short script.

Artifact Drops are publication-only. This archetype is the clearest publication-only mode in the corpus. Repositories ship packaged archives or binaries rather than source code, with a median lifespan of about seven minutes from first to last commit. The content reads like a malware museum: WannaCry, NotPetya, Conti, Babuk, Zeus, Citadel, and Mirai variants sit alongside cracked commercial RATs. One example is `AJMartel/MeGa-RAT-Pack`, with 30 cracked RATs in 10 commits.

Quiet Iterative is the engineering branch. This is the one archetype in the corpus that looks like serious software engineering. Repositories iterate over a median of twelve active weeks with little public coordination, and 83% of their pull requests are written by

people rather than bots, the highest share of any archetype. Rootkits, which are among the most technically demanding malware types to build, make up 6.7% of this cluster, the highest concentration in the dataset.

Discussion-Driven repositories are small but loud. This archetype is where the malicious repository ecosystem is most visible as user activity. Discussion-Driven repositories are only 2.9% of the corpus, yet they hold 24.9% of all issues and 72.9% of all pull requests in the dataset. They also carry nearly all of the corpus-wide IOC-pattern matches recovered by regex from repository text and source files: 1 Discord webhook URL with an embedded bot token, 8 Discord invite links, 5 Telegram channel URLs, and 37 IPv4 addresses. We do not probe or interact with any extracted endpoint, so we do not claim that any address is live or operationally active. The clearest example is **Blank-c/Blank-Grabber**, a Discord info-stealer that holds 294 issues by itself, with threads mixing feature requests, runtime errors, and victim complaints.

***Observation 8.** The lifecycle split is sharp: 2,887 repositories (80.24%) are ephemeral, while 711 repositories (19.76%) are sustained. The small sustained slice contains most observable development and user interaction.*

3.4 User Communities and Participation

How is user activity organized around the 711 sustained repositories? We answer this question by separating user participation into three roles, building one bipartite repo-user graph per role, and detecting communities within each.

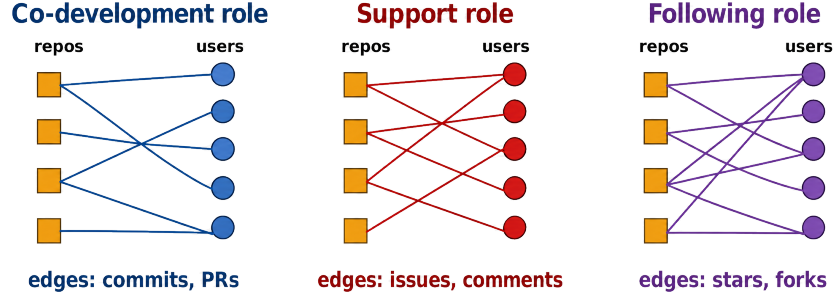


Figure 3.6: The three-role decomposition. Each role uses the same repository universe but keeps a different class of repo-user interactions.

3.4.1 Three role views

We construct three repo-user bipartite graphs, one for each participation role, as shown in Figure 3.6.

- a. **Co-development role:** commits and pull requests. Code-level contribution.
- b. **Support role:** issues and comments. Help-seeking, debugging, complaints, operational discussion.
- c. **Following role:** forks and stars. Copying, bookmarking, attention.

We use *user* for any GitHub account that interacts with a repository through any of these six event types. Across the three role graphs on the sustained scope, we observe 51,194 distinct users.

3.4.2 Graph construction and communities

Before constructing each role graph, we apply three cleaning steps: we normalize login strings, remove known automation accounts such as Dependabot, and remove owner-on-owner-repository events (not external participation). We run community detection on each role graph independently using Leiden modularity optimization [105] at a fixed seed. A

Table 3.3: Sustained-scope role coverage after cleaning. A strict community has at least three users and three edges.

Role	Repos	Users	Communities	Community users	Community repos	Med./max users
Co-development	659	1,100	80	484	94	4/30
Support	213	2,829	87	2,711	120	12/213
Following	565	48,418	60	48,342	501	405/3,796

community is a *strict community* if it has at least three users and three edges. Table 3.3 reports the resulting coverage. The three roles differ sharply in scale: follower contains 48,418 users, roughly 44× more than co-developer’s 1,100, and the same gap holds inside strict communities.

3.4.3 Findings per role

Co-development is local. The co-developer role gives the strictest test of cross-repository collaboration: do users commit code or submit pull requests to more than one malicious-intent repository? The answer is rarely. Of the 80 co-developer strict communities, 70 contain only a single repository and 10 span multiple repositories. After a manual audit of the 10, only one represents a genuine cross-owner pair: `UndeadSec/EvilURL` and `shakenetwork/SocialFish`, both phishing tools. The other nine candidates are fork chains, in which the same code is re-uploaded by a different account, or same-owner portfolios. Genuine cross-team code collaboration is essentially absent.

Support is broader, but still operational. The supporter role contains more users and supports larger single-repository communities. The largest is `Blank-c/Blank-Grabber`, a Discord info-stealer, with 213 distinct supporter users and 968 supporter events. Multi-

repository supporter communities also exist, but they are typically formed by users asking similar setup or troubleshooting questions across related tools rather than by coordinated collaboration. One recurring supporter user, for example, appears across phishing, keylogger, and wireless-attack repositories asking how to run them or how to fix errors. This pattern is cross-tool support-seeking, not cross-tool development.

Following is the widest role, and groups repositories by malware type. The follower role dominates the ecosystem in user count, and it is the only role in which multi-repository communities are the norm: of the 60 follower strict communities 40 span more than one repository. The repositories in each multi-repository community share a common malware type. The largest community contains 3,796 users following 65 rootkit and backdoor repositories, such as `mempodippy/vlany`, `unix-thrust/beurk`, and `EmpireProject/EmPyre`. Other large follower communities form around Android attack tools, DDoS toolchains (`R00tS3c/DDoS-RootSec`, `HyukIsBack/KARMA-DDoS`), botnets, and stealers. Individual engagement remains shallow even in these large communities: the average user touches close to one repository.

3.4.4 The participation pyramid

Across the three role graphs the sustained corpus contains 51,194 distinct users. Figure 3.7 shows the share of these users covered by each role. The follower role covers 48,418 users, supporter covers 2,829, and co-developer covers 1,100. The roles are not disjoint, so the shares reflect per-role coverage rather than a partition; only 75 users (0.15% of the total)

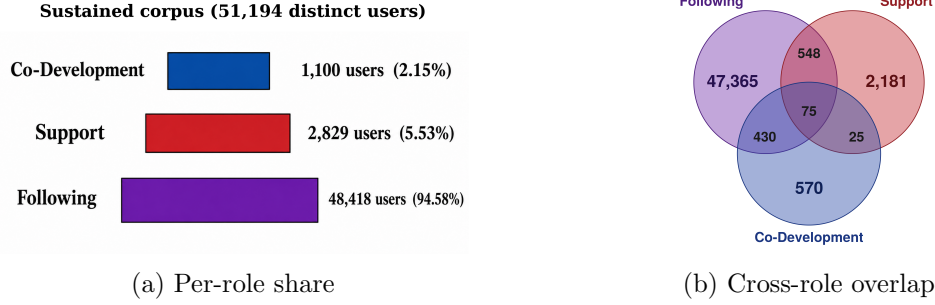


Figure 3.7: Per-role structure of the sustained corpus. (a) Share of the 51,194 distinct users who participate in each role; roles overlap, so shares do not sum to 100%. (b) Venn decomposition: only 1,078 users sit in two or more roles, and just 75 in all three.

appear in all three. The open-source path from watching to filing issues to contributing code is extremely narrow in this ecosystem.

The sustained subset also captures most user activity in the corpus. Although sustained repositories represent only 19.76% of the 3,598 malicious-intent repositories, they account for approximately 80.6% (51,194 users) of all users observed across roles in the full corpus (63,496 users). The ephemeral branch adds many repositories but little user structure.

Observation 9. *The user ecosystem is wide but shallow: follower contains about 44× more users than co-developer, and only 0.15% of users participate across all three roles.*

3.5 Discussion

In this section, we discuss the implications, design choices, and limitations of MAGNET.

a. What is the overall picture of this sub-ecosystem? Malicious-intent repositories on GitHub are usually published and watched more than they are collaboratively developed. Some are maintained and discussed, but deep activity is confined to a small sustained minority and is usually local to one repository.

b. Why does the archetype taxonomy matter? The taxonomy gives a practical way to prioritize the corpus. Treating all 3,598 repositories equally would waste attention on inactive source snapshots and artifact drops, while the 103 Discussion-Driven repositories are a more useful monitoring target because they concentrate issues, comments, pull requests, and operational context.

c. Why does the three-role model matter? A collapsed social graph would blur three different roles. Co-development communities measure code work, supporter communities expose operator questions and victim reports, and follower communities expose curation and reputation signals. The last role is especially relevant for campaigns that weaponize stars and forks [99].

d. What does this imply for platform monitoring? Monitoring should not rely on a single signal. A high-star repository may be a following object rather than an engineering hub, and a high-issue repository may be operationally active with little code work. The three-role separation makes these distinctions measurable.

e. How should this work be released responsibly? The goal of MAGNET is measurement and defense, not redistribution of malware. A public release should prioritize repository identifiers, derived labels, metadata, aggregate statistics, and analysis code, subject to institutional and platform policies. Runnable malware artifacts should not be republished.

f. What are the main limitations? The corpus starts from repositories found by prior work and a conservative intent filter, so the results should not be read as covering all malicious activity on GitHub or non-GitHub channels. Malicious intent is inferred from observable repository and author evidence, not from private motivation. The LLM agent

can still make mistakes; we mitigate this with a strict rule and blinded manual validation. Identity resolution is imperfect because a single human may use multiple accounts, which can under-count or over-count multi-repository overlap. Some repositories and accounts may also disappear after collection, which we treat as availability signals rather than as evidence of platform policy.

3.6 Related Work

We group prior work in three directions: detection, actor profiling, and code-centric malware analysis.

Detecting malware repositories. SourceFinder is the foundational malware-repository study on GitHub, using supervised learning over repository metadata [85]. Rep2Vec extends this with heterogeneous-graph embeddings [82], and recent work tracks malicious code reuse across GitHub, StackOverflow, and underground forums [100]. These works answer the discovery question: which repositories contain or reuse malicious code? MAGNET asks a different question: it separates malicious intent from benign security framing and studies how the resulting repositories behave as projects.

Profiling actors and communities. A second line studies the actors behind malicious or security-relevant activity. HackerScope analyzes malware authors on GitHub by combining GitHub activity with security-forum evidence [49]. Other cybercrime and threat-intelligence studies use forums and online discussions to identify actors, warnings, and underground communities [91, 80]. The closest work to our network analysis is HackerScope, which

merges follower, fork, contribution, and comment relations into one weighted graph. In contrast, we keep contribution, support, and audience separate, allowing us to show that interaction levels and communities differ by role.

Analyzing malware source code. A third line studies malware as software. Calleja et al. analyze decades of malware source code from a software-metrics perspective [9] and later quantify complexity and code reuse in malware development [10]. This line is close in spirit because it treats malware as a software artifact. The difference is the unit of analysis. These studies focus on the code and its properties. MAGNET focuses on public repository behavior: lifecycle, maintenance, coordination, audience, and user participation on GitHub.

3.7 Conclusions

We propose MAGNET, a software-engineering framework for studying malicious-intent repositories on GitHub. The framework identifies a high-precision corpus of 3,598 repositories from 16,726 candidates, organizes them into a three-level hierarchy of engineering archetypes, and models user participation through three roles: co-developer, supporter, and follower.

Our central finding is that the ecosystem is wide but shallow. Most repositories (80.24%) are ephemeral one-time uploads, while the sustained minority concentrates almost all observable engineering activity. User activity around this sustained minority is concentrated in the follower role: 48,418 users appear through forks and stars, compared with only 1,100 users in co-developer. The ecosystem is much more visible as an audience surface than as a co-development space.

We provide MAGNET as a building block toward studying malicious software ecosystems on public software-hosting platforms through a software-engineering lens. Future work can adapt MAGNET framework to other platforms where code, reputation, and social interaction co-occur.

Chapter 4

GeekMAN: Disambiguating Usernames Across Platforms

How can we identify malicious hackers participating in different online platforms using their usernames only? This is the question that motivates our work. Establishing the identity of a user across online platforms (e.g. security forums, GitHub, social media) is an essential capability for tracing malicious hackers. First, hackers with visible online personas often lead major cyber-criminal activities [90]. Second, these hackers are active and visible on many online platforms including specialized security forums and popular platforms like GitHub [49]. In fact, some of these platforms harbor malicious activities to the point that they are forced to shut down [26]. Although a hacker could pick arbitrary names, they often use the same or similar usernames as this helps them establish an online “brand”. As a result, hackers (a) adopt unusual names, and (b) use them fairly consistently with

only minor changes across different platforms. For example, a username of interest could be *w33dgod*, which we may want to match with *godweed* (both are real usernames).

Problem. The problem we address here is the following: given two usernames, how can we determine if they are likely to belong to the same user? As our focus is tracing hacker activity, we focus on **technogeek** usernames, which we define as usernames with: (a) technical jargon, (b) slang and unconventional use of letters and characters, and (c) multiple parts. These types of usernames seem to be used by malicious hackers, but also by tech-enthusiasts, gamers, etc. For example, *IAmBlackHacker* on GitHub and *B14CKH4K3R* on Facebook¹. We refer to this kind of *obfuscation* using letters and digits in unusual ways as **slangification**. Many of these usernames have multiple parts, which we refer to as **chunks**. Traditional string matching and edit distance techniques have difficulty matching these types of usernames. Here, we impose an additional challenge: we do not use other types of information, such as demographic attributes, context, or social connections, which could help refine the matching accuracy. Our framing of the problem can be seen as an *essential building block* of a broader solution. Thus, our goal is to push the username-level matching to its limit: how far can we go with just usernames?

There has been relatively little work on the problem as we define it here. In particular, we find that most of the previous works: (a) focus on the popular social-media usernames, (b) rely on training data, and (c) use string matching without following a human-like interpretation, such as decomposing the username into meaningful chunks. Specifically, many methods treat the username as a string, and use features such as the frequency of bi-grams

¹All examples here are real usernames. The GitHub malware author *IAmBlackHacker* refers to a Facebook page as a blog in the profile information. The Facebook page claims Varanasi, India as its location, so we suspect that *black* must refer to black hat hacking.

and tri-grams. Overall, we can group the previous efforts into three large families depending on their primary focus on: (a) concentrating on username similarity [81, 114, 110, 61], (b) leveraging user profile attributes, such as demographic information (e.g. gender, education, job title) [107, 37, 117, 74], and (c) combining multiple dimensions of information, such as the user-generated content, including topics and linguistic style, and in-platform social connections [62, 63, 118]. As we explain later, we compare our approach against a set of state-of-the-art username similarity algorithms [110, 61]. We discuss previous works in Section 4.6.

As our key contribution, we propose GeekMAN, a systematic approach for linking technogeek users across platforms. Our approach is inspired by human cognition: it attempts to emulate how a human will try to disambiguate this type of username, such as *IAmBlackHacker* and *B14CKH4K3R*, which we mentioned above. The key novelty of our work consists of the development and integration of three capabilities: (a) **deslangification**, which de-obfuscates slang and “geeky” naming conventions, (b) **chunkification**, which decomposes usernames into meaningful chunks, which leads to one or more lists of chunks, and (c) **comparison**, which considers all the lists of chunks to calculate the similarity between two given usernames. We introduce our metric of Similarity Score, *SimScore*, and we use the Similarity Score Threshold, *SimT*, to select username pairs that are a likely match. We deploy our approach on 1.8M usernames from three different types of online platforms: (a) five popular hacker-rich security forum users, (b) 7.3K malware authors from GitHub, and (c) three mainstream social media platform users, which we use as reference.

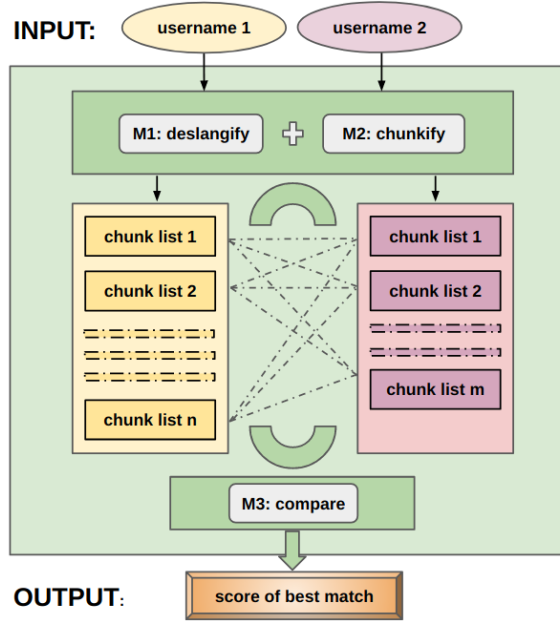


Figure 4.1: GeekMAN calculates the similarity score for a pair of usernames focusing on technogeek users. Our approach tries to emulate human interpretation by combining the (a) Chunkification, (b) Deslangification, and (c) Comparison modules.

The key results are summarized below.

a. Technogeek usernames use slang and chunks extensively. Technogeek forums have 1.5-6 times more usernames with multiple in-between digits or symbols than social media platforms (Section 4.2), and 60-70% of their usernames decompose into at least four chunks, with roughly 20% exceeding six.

b. GeekMAN outperforms prior approaches especially on technogeek usernames. Focusing on technogeek usernames, we find that our approach identifies matches with 86.0% Precision and 72.6% Relative F1-score (which we define later). By contrast, two prior approaches exhibit 76.0% and 47.4% Precision with 42.9% and 57.1% Relative F1-score, respectively. Our approach outperforms prior methods even with usernames of common social networks.

c. GeekMAN finds 6327 forum users who likely own malicious repositories on GitHub. As an indicative use of our approach, we match security-forum users to malicious GitHub authors [85]. We find 6327 such matches with a similarity score threshold of $SimT = 0.7$. Manual investigation of a set of randomly selected matched users shows that often the forum users proudly point to their own GitHub profile, which validates the match.

d. GeekMAN allows for balancing the Precision-Recall trade off. Depending on the need of the study, our approach can be easily tuned to favor higher Precision or higher Recall. The 6327 likely matches between forum and GitHub users for $SimT = 0.7$ become 1958 matches for Similarity Score Threshold $SimT = 0.9$. Note that only 260 matches correspond to identical strings: our method adds value beyond the obvious matches. We find that 57.7% of these 1958 matches use digits/symbols in their usernames as opposed to only 16.2% of among the 260 exact matches.

e. An online platform, datasets, and a “deslangification dictionary”. To facilitate further research, we provide: (a) an online tool, as shown in Figure 4.7, (b) our datasets, (c) our ground truth, and (d) a slang-translation dictionary. Currently, our slang-translation dictionary has 5.8K entries consisting of slang words and their counterparts. We have shared our source code on GitHub [68].

Our work in perspective. Our approach is a fundamental building block for tracing users across different platforms with an emphasis on technogeek usernames. We revisit its practical impact and limitations in Section 4.5. In the same section, we also discuss the issue that a matching pair of usernames does not necessarily mean that they belong to the same user.

4.1 Background and Data

In this section, we provide some background, explain the motivation of our approach, and discuss the dataset in detail.

A. Malicious hackers use technogeek in usernames. Hackers and other malicious users often participate in various public platforms, including specialized discussion forums, technical forums, and software platforms like GitHub. Their main goals seem to be: (a) establishing an online brand, and (b) boasting of their accomplishments [49, 50]. For example, we have seen posts by hackers to advertise their services, claim the credit of successful hacking activities, and create communities of like minded people [51].

Often they use the same username consistently across platforms, but sometimes they vary the spelling or structure of the name or the order of the parts of their names. They sometimes choose different username spelling via transformation of their original alias in order to establish a branding or hide their own persona in the forum. We refer to this username spelling transformation as **slangification**. As a noteworthy example, an FBI most wanted cybercriminal, having alias *ha0r3n*, was found to have a GitHub profile named *wo4haoren*. In 2020, FBI listed another most wanted hacker named *Behzad Mohammadzadeh*, with alias *Mrb3hz4d*, who was charged for defacing a number of websites [21].

B. Datasets. Our dataset contains nine different online platforms spanning across three broad categories: security forums, open-source software platforms, and social networks. Table 4.1 displays a summary of them.

a. Security forums: This category comprises five security forums: Garage4Hackers(GH), Offensive Community(OC), RaidForums(RF), Multiplayer Game Hacking(MP), and Hack-

Table 4.1: Summary of our datasets: security forums, GitHub, and social media.

Category	Platform	Abbr.	Users
Security forums	Garage4Hackers	GH	865
	Offensive Community	OC	11,371
	RaidForums	RF	44,107
	Multiplayer Game Hacking	MP	507,945
	Hack Forums	HF	659,672
Software platforms	GitHub	GT	7,389
Social media	Facebook	FB	163,037
	Twitter	TW	196,534
	Google+	GP	280,318

forums(HF) [23, 79, 83, 73, 42]. The data contains posts and threads of 1.2M users between 2005 and 2021, gathered through our automated crawler and shared by the Cambridge Cybercrime Centre [11].

b. The GitHub software platform: Interestingly, hackers share malware source code in public platforms, such as GitHub. We consider 7389 GitHub (GT) authors, who were identified to have at least one repository with malware source code [85, 96].

c. Social media: We also consider 639K usernames [36] from three popular social networks: Facebook(FB), Twitter(TW), and Google+(GP) (the popularity of Google+ did not last). Among them there are 49K pairs of usernames that belong to the same user, which we use later in our evaluation. We use this dataset to compare the username patterns in

typical social networks against the patterns in our technogeek platforms, and to evaluate the performance of GeekMAN in regular usernames. For the rest of the paper, we use the term technogeek to refer to the first two categories: security forums and GitHub.

Validation and ground truth. Note that the social media dataset consists of verified accounts owned by the same users. So we use this to partly evaluate our algorithms. However, given our focus on technogeek forums, we describe our validation approach in Section 4.4.

4.2 Quantifying Technogeek Usernames

In this section, we study the usernames in technogeek forums, which are likely to be visited by hackers. The goal is to understand the username selection patterns and conventions in order to inform our approach. Overall, we find that the usernames in technogeek forums are different from those in general purpose platforms, such as social media.

a. Technogeek forum users use symbols and digits more frequently than social media users. We measure the number of usernames that contain symbols and digits to test our intuition that technogeek usernames contain these characters more frequently and in more unusual ways compared to social media usernames. We plot the distribution of the users in different platforms who use digits in between two alphabetical characters in their usernames in Figure 4.2. We see that around 17~37% of technogeek forum usernames have multiple digits in between letters of their usernames, which is at least 1.5-6 times compared to social media users. This indicates that the technogeek users are more likely to use this type of usernames. For example: *n1nj4sec* and *z3r0d4y* contain two and three digits in

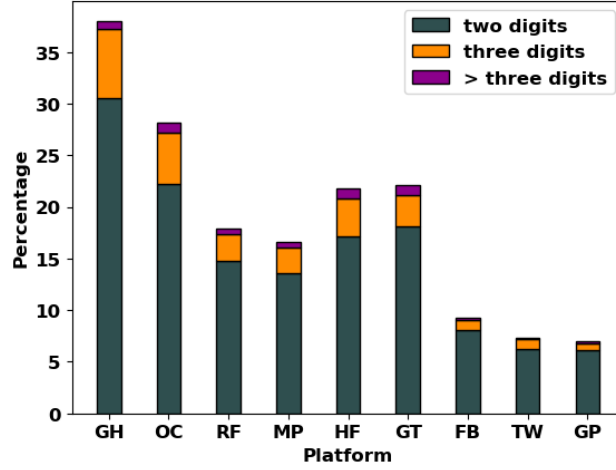


Figure 4.2: Distribution of percentage of population who use digit in between letters multiple times. An example username containing two digits in between letters: *g3ntl3man*.

between letters, instead of *ninjasec* and *zeroday*. Furthermore, we find that 22% of the GitHub users in our dataset, indicated as GT in the plot, also demonstrate this type of behavior. Recall that our GitHub users have been identified as malware producing authors, so this percentage will most likely differ if we consider GitHub users at large. By contrast, this behavior is much less pronounced in social media platforms, such as Facebook. We find similar trends for using symbols during the investigation.

b. We find 60-70% of usernames with multiple chunks. We want to quantify how frequently usernames consist of multiple parts, which we refer to as **chunks**. We use the Chunkification method that we describe in the next section on all the online platforms in our dataset. For each username, we consider a list of all possible chunks, which we could find from different chunkification approaches. In Figure 4.3, we plot the CCDF (Complementary Cumulative Distribution Function) curve of the total number of chunks found per username. We find that approximately 60-70% of the users on each of the online platforms contain 4 or

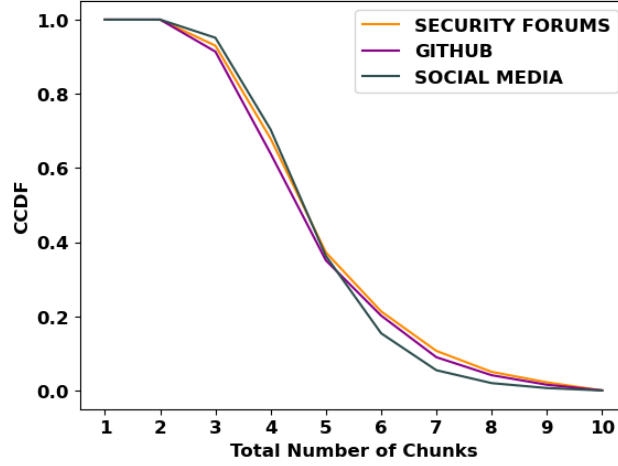


Figure 4.3: The CCDF of total number of chunks found per username across different types of online platforms. An example username: *sniper7kills* having three chunks $\{sniper, 7, kills\}$.

more chunks in their usernames, irrespective of technogeek and social media platforms. We also notice that the number of chunks per username seems consistent across all platforms, although the frequency of more than 5 chunks is slightly higher among the technogeek forums.

In an effort to get a deeper understanding, we also investigated the username chunks. We find that general users prefer proper English words and names in their chunks. By contrast, technogeek forum users transform the spelling of English words in their usernames by replacing one or more letters with digits/symbols. Our analysis concludes that the percentage of technogeek users who use slangification is 2-8 times higher compared to social media users. We illustrate the most commonly slangified English words used by technogeek forum users via a word cloud shown in Figure 4.4.



(a) Technogeek forums



(b) Social media

Figure 4.4: The word-cloud that shows popular slangified words used at least 10 times as usernames or as parts of usernames in technogeek forums (200 words) than social media (6 words only).

4.3 Proposed Method

The goal of our method is to determine the similarity score of two given usernames. The inspiration behind GeekMAN is the emulation of a human interpreter. The key idea is to de-obfuscate the usernames (if possible), decompose them into several possible lists of chunks, and finally compare the lists of chunks to calculate the similarity score.

Our approach consists of three main modules: (a) Deslangification, (b) Chunkification, and (c) Comparison. We provide a conceptual overview in Figure 4.1 and demonstrate its operation for usernames *z3r0c00l* and *COOL_zERO* in Figure 4.5.

a. Deslangification module: This module aims to deslangify a given username by: (a) identifying likely use of slang, and (b) translating the slang into regular words. One such example is: *z3r0c00l* which most likely refers to *zerocool*.

b. Chunkification module: This module takes a username and produces one or more lists of *chunks*. We use the term **L** to refer to each such list and the term **Bag** to refer to

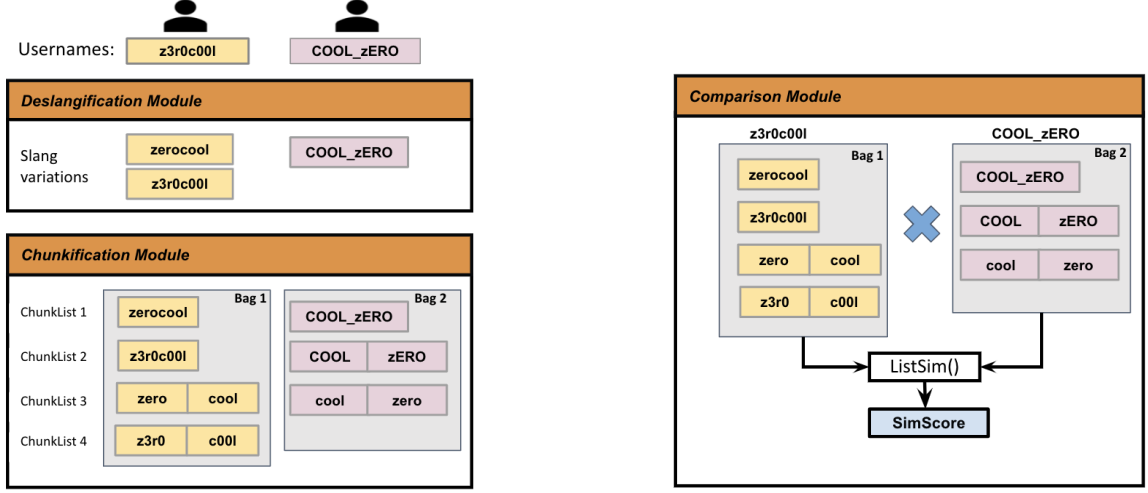


Figure 4.5: An example of how the modules of our approach will handle a pair of usernames: *z3r0c00l* and *COOL_zERO*.

all the lists of chunks for a username. Table 4.2 provides a list of our terminology. Note that in its most general case, the Chunkification module can provide more than one possible ways to chunkify the username. E.g. *mr-satan1* can be decomposed into: $\{mr, -, satan, 1\}$, and $\{mr, -, satan1\}$, since *Satani.* could be a slangified version of someone’s last name. In fact, we want to generate as many as possible plausible lists of chunks to ensure the highest possible similarity.

c. Comparison module: This module takes as input two Bags with lists of *chunks*, one for each username, and returns the highest similarity score among all possible pairs of lists. In more detail, it consists of functions that calculate the similarity: (a) between two *chunks*, (b) between two lists of *chunks*, and (c) between two Bags of lists of *chunks*.

4.3.1 Our Approach in Detail

We now describe our method in more detail by discussing our algorithmic choices and the related challenges. For convenience, we list key terms and functions in Table 4.2.

Table 4.2: Definition of terminology for GeekMAN approach.

Terms	Definitions
<i>alpha</i>	A lowercase or uppercase English letter.
<i>num</i>	A numerical character.
<i>sym</i>	A non-alphanumeric character.
<i>char</i>	A character: <i>alpha</i> , <i>num</i> , or <i>sym</i> .
<i>chunk</i>	A string of <i>chars</i> .
L	A list of <i>chunks</i> .
Bag (<i>u</i>)	A set of L generated from username <i>u</i> .
<i>slangChar</i>	A single <i>num</i> or <i>sym</i> which is known to be used in place of an <i>alpha</i> .
<i>slangCharMap()</i>	A function that maps a potential <i>slangChar</i> to <i>alpha</i> characters. E.g. “3” to “e” and “3” to “s”.
<i>TokenDict</i>	A dictionary of English word/name phrases.
<i>ChunkSim()</i>	A function that returns the similarity between two <i>chunks</i> .
<i>ListSim()</i>	A function that returns the similarity between two L.

A. Maximizing the likelihood of a match. The task of reverse engineering the naming habits of users is a challenging problem. The overarching goal is to ensure that we find a possible match, even in the face of obfuscation. The two modules of deslangification and chunkification can be used synergistically and in conjunction.

To have the broadest possible coverage, we can consider the following approaches:

1. we do deslangification and then chunkification
2. we do chunkification and then deslangification
3. we keep the initial username as is as a one-chunk list
4. we keep the deslangified username as one-chunk list
5. we filter out unlikely lists of chunks (optional)
6. we apply chunkification iteratively on chunks

Note that the best results are derived from the first sequence as explained in Section 4.4.

B. Deslangification module. In this module, we try to de-obfuscate the username to identify any available slangified *chunk*. First, we create *slangCharMap()*, a function that maps a potential slang character to a letter following the common technogeek conventions based on our observations and commonly reported usage [112]. Then, we search for potential slang characters in the username, and if any are found, we replace them with the corresponding letter character found from *slangCharMap()*. As an example, username *th3m4lw4r3* can turn into *themalware* and username *z3r0s4mur41* into *zerosamurai*. We notice that digit 4 can be translated as both a and r. In addition, digit 4 does not necessarily have to transform into a letter. Thus, we obtain a plethora of potential deslangified versions from one username along with the original username string.

C. Chunkification module. We perform chunkification on a given username based on different criteria and create a Bag which is a set of lists of *chunks*. We consider four different chunkification criteria emulating four different aspects of username naming behavior. The

algorithmic challenge is to strike the following balance. On the one hand, we want to extract as many as possible *reasonable* lists of *chunks* to maximize the possibility of similarity. On the other hand, doing all possible chunkifications without any semantic cues would create a combinatorial explosion of lists. We address this problem with the following methods.

a. Symbol-based chunkification: Some symbols are used as delimiters to chunkify usernames. Typically delimiter symbols are the underscore, the space and the dot. In Section 4.2, we discussed the idea of using symbols in usernames. We create L , a list of *chunks* using the symbols present in the username. An example is *COOL_zERO* which we can split into $\{COOL, zERO\}$. Note that we could of course keep delimiter symbols as chunks in the list, but we consider that they would rather impede than help matching accuracy.

b. Digit-based chunkification: Numbers are sometimes used to separate words, and we use this here as well. Digits in a username can be either something telling about the user or just a combination of single/ multiple random digits. For example, in one case a user may choose to use digits related to her personal information, while in another case a user may use random digits simply to make her username comply with certain platform requirements. We search for numbers in the username, and if any are found, we chunkify the username accordingly. An example of digit based chunkification can be: *sniper7kills* will be split into $\{sniper, 7, kills\}$.

c. Capitalization-based chunkification: Capital letters can also provide cues for chunks. Capitalization is often used by users to specify multiple parts of their usernames. For example, in the username *ObscureCoder* is reasonable to assume that the user has com-

bined *Obscure* and *Coder*. In a more challenging example, *T0x1c V3n0m* can be split into $\{T0x1c, V3n0m\}$ which could lead to $\{Toxic, Venom\}$. Note that in both these examples, we leverage the appearance of commonly-used English words, but if users use obscure geographical, or regional names, things can become more complex.

d. Token-based chunkification: We also propose an approach to detect chunks in the absence of cues, finding potential words that could be “hiding” in the username. For instance, *thegreathacker* corresponds to $L = \{the, great, hacker\}$.

There are many different ways to identify words in a string [43]. We follow the approach below. We start from the end of the string and consider letters until we find a word that exists in our *TokenDict*, a dictionary of English word/name phrases. In our example that word would be *hacker*. We then create two parallel approaches: (a) we repeat the same process on the string, having removed *hacker*, and (b) we continue to see if the word *hacker* is part of a longer word. At the end of this process, we have several lists of *chunks*.

D. Comparison module. Given two Bags with lists of chunks from usernames u_1 and u_2 , we calculate the highest similarity score among all possible comparisons between the lists of *chunks*. We are now ready to provide a more formal description of our approach while an intuitive visualization is provided in Figure 4.5. Given usernames u_1 and u_2 , earlier modules produce two sets of lists of *chunks*, $\text{Bag}(u_1)$ and $\text{Bag}(u_2)$, respectively with N and M , the number of lists in each Bag.

$$\text{Bag}(u_1) = [L_1^1, L_1^2, \dots, L_1^N]$$

$$\text{Bag}(u_2) = [L_2^1, L_2^2, \dots, L_2^M]$$

We use three similarity functions: (a) $ChunkSim(c_1, c_2)$ for *chunks* c_1 and c_2 , (b) $ListSim(L_1, L_2)$ for two lists L_1 and L_2 , and (c) $SimScore(Bag_1, Bag_2)$ between two bags Bag_1 and Bag_2 . The additional complexity is that at the list and Bag level, we need to iteratively calculate many possible potential matchings. We present a visualization of our approach in Figure 4.5 using the real usernames: *z3r0c00l* and *COOL_zERO*.

a. Similarity of chunks. We estimate similarity between chunks of strings. There are many string matching algorithms, and our approach could use any of them. We select the Levenshtein method [58] which is widely used in text processing [27]. We use the term $ChunkSim(c_1, c_2)$ to refer to this function.

b. Similarity of lists of chunks. Comparing lists of chunks is slightly more complex, as we need to identify the most likely match between the *chunks* of the two lists L_1, L_2 (note that for clarity we drop the superscript in the notation). We use the term $ListSim(L_1, L_2)$ to refer to this function. There are many different algorithms that we can use to compare similarity of unordered lists that vary in efficiency and computational complexity. Our approach could use any such function. In our current implementation, we use the Monge-Elkan method [72] which is found to perform well consistently across many scenarios and data types [7] with a polynomial computational complexity of $O(|L_1| |L_2|)$. Intuitively, the method iterates through the elements of the first list and identifies the highest similarity with any element in the second list. The final value is the average chunk similarity. Formally, the method calculates the similarity for the two lists as follows:

$$ListSim(L_1, L_2) = \frac{\sum_{i=1}^{|L_1|} \max_{\forall j} ChunkSim(L_1[i], L_2[j])}{|L_1|}$$

where $1 \leq j \leq |L_2|$ and $L_1[i]$ and $L_2[j]$ are the i -th and j -th *chunks* of each list.

This method hides a subtle point. The function is sensitive to the order of the arguments: $ListSim(L_1, L_2) \neq ListSim(L_2, L_1)$. Therefore, one could consider three approaches. We can consider : (a) $ListSim(L_1, L_2)$, (b) $ListSim(L_2, L_1)$, or (c) considering both “directions”, $ListSim(L_1, L_2) + ListSim(L_2, L_1)$. In the results that we show here, we use one direction with the longest list as the first argument, namely, assuming $|L_1| \geq |L_2|$, $ListSim(L_1, L_2)$.

c. Similarity of Bags of lists. This is the last step in assessing the similarity between two usernames, which are represented by their Bags. We use the term Similarity Score $SimScore(u_1, u_2)$ where u_1 and u_2 are the two usernames. We do the comparison exhaustively: each list in the Bag of one username is compared with each list of the other username using the list similarity function $ListSim()$ from above. The Similarity Score is the maximum list similarity over all list pairs L_1^n, L_2^m as follows:

$$SimScore(u_1, u_2) = \max_{n,m} ListSim(L_1^n, L_2^m)$$

where $1 \leq n \leq |Bag(u_1)|$, $1 \leq m \leq |Bag(u_2)|$.

4.3.2 Computational complexity

In our experience, the complexity was not prohibitive with most usernames having a single-digit number of lists, and each list having single-digit number of chunks. The computational complexity of the Comparison module is bounded by $O(|B_{max}|^2 |L_{max}|^2)$, where $|B_{max}|$, $|L_{max}|$ are the largest Bag and list L length, respectively. In practice, in matching 2M pairs of usernames, we find that it takes approximately 5 minutes in a Ubuntu 20.04.5 Linux machine with 64GB RAM and 8 CPU cores.

4.4 Experiments and Evaluation

We evaluate our approach by comparing it against state of the art methods, and we demonstrate its usefulness by identifying forum users who could be malicious hackers on GitHub.

4.4.1 Evaluating GeekMAN

Here, we present our evaluation study and discuss our ground truth, comparison metrics, and the baseline algorithms. At a high level, our experiment aims to match a set of users from a *source* forum with a set of users from a *target* forum.

A. Baseline algorithms. The purpose of conducting an experiment is to determine how GeekMAN compares to two baseline state-of-the-art username matching methods: Wang-16 [110] and UISN-UD [61].

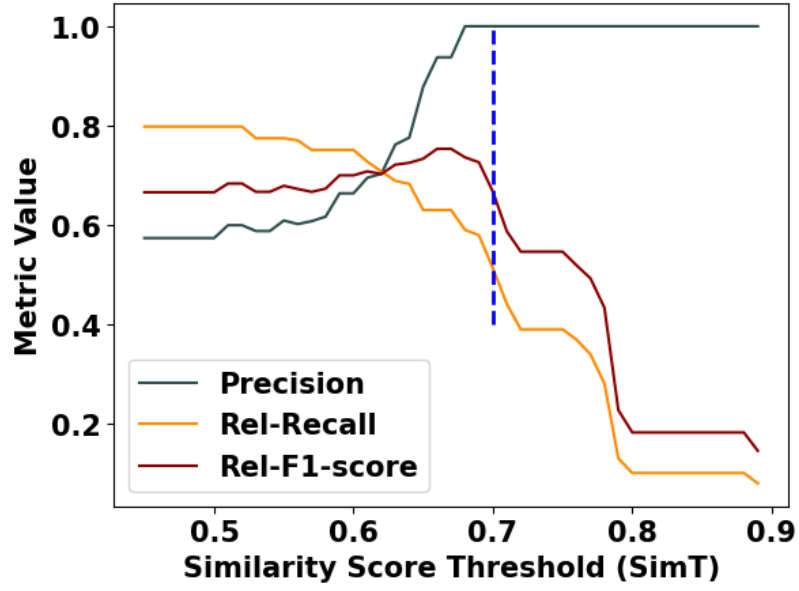
a. Wang-16: This method extracts content features (e.g. 2-grams), and pattern features (e.g. letter-digit, date) from the usernames. Then, a vector based modeling is used to compute the cosine similarity of the vectors of features from usernames.

b. UISN-UD: This method exploits the information redundancies that can be available in a pair of usernames used by the same user. It computes features from different string comparison metrics, such as common substring, common subsequence, and edit distance, which are used in their classifier.

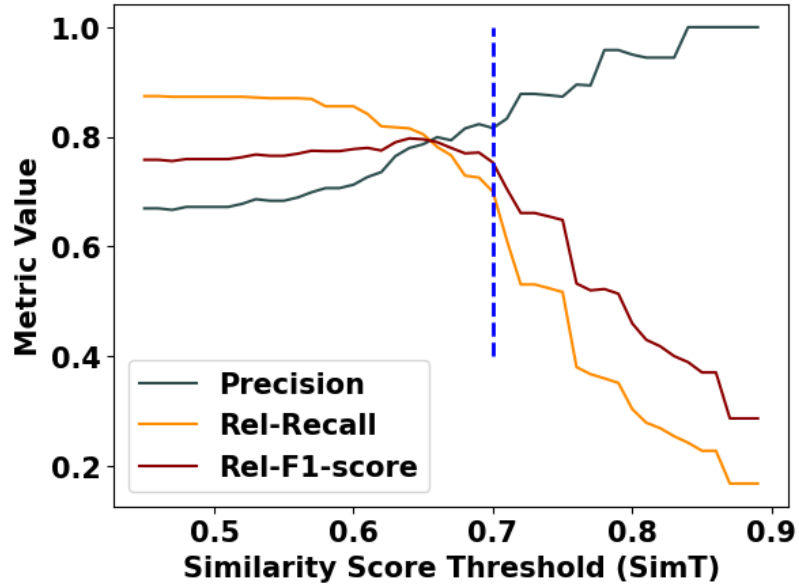
The major difference of GeekMAN with them is the use of deslangification and chunkification of the technogeek usernames, assuming such properties exist in the comparing usernames.

B. Experimental process and setup. We conduct two experiments focusing on (a) technogeek and (b) regular usernames. **Experiment-T: technogeek usernames.** We find matches between usernames from a *source* forum within a *target* forum. Specifically, we conduct two such experiments: (a) Garage4Hackers (GH) as *source* and Offensive Community (OC) as *target*, (b) Garage4Hackers (GH) as *source* and RaidForums (RF) as *target*. Given our focus on technogeek names, we selected Garage4Hackers as *source* since it exhibits a higher level of slangified and chunkified usernames, based on our analysis of these forums (see Section 4.2). On the other hand, the *target* forums were picked randomly. To focus on technogeek usernames, we selected usernames from our source forum with either 3 or more *chunks* or slangification. This way we obtain the **D_All** dataset, which is roughly 10% of the Garage4Hackers forum (on purpose small to enable validation as we discuss below). We also divide D_All into **D_Multi** (3 or more chunks) and **D_Slang** (slang conventions) datasets, in an effort to investigate the interplay between multi-chunk and slangified usernames such as $\{thegreathacker, T0x1cV3n0m\}$. We use GeekMAN and baselines to match each user in D_All with the most likely matching user in the *target* forums.

Experiment-R: Regular usernames. We evaluate our algorithms on regular usernames using a subset of the social media datasets of Table 4.1, which have confirmed user correspondences across the social platforms. We create a dataset **D_Social** of 98K username pairs from Google Plus (GP) and Twitter(TW): (a) 49K pairs that belong to the same user, and (b) 49K pairs that (most likely) do not, selected randomly with a bias toward Levenshtein similarity ≥ 0.3 to make the classification harder.



(a) D_Multi: Precision, Rel-Recall, Rel-F1-score.



(b) D_Slang: Precision, Rel-Recall, Rel-F1-score.

Figure 4.6: The effect of $SimT$ on the performance of GeekMAN. The blue vertical line at 0.7 marks the point that finds a reasonable trade-off with a slight emphasis on Precision.

C. Validation and ground truth. There exists ground truth for the Experiment-R, but there is no available ground truth for Experiment-T, so we need to establish our own. We resort to sampling and manual verification. The algorithms find the best matching user in *target* forum for the users in D_All. We recruit four domain-expert computer scientists to manually label each of these possible matches as a match or mismatch. To increase the reliability of our ground truth, we ask the annotators to match a username pair only if they were *certain* they belong to the same user based on usernames only. We only consider a username pair as a verified match if at least three annotators agree it is. Here we focus primarily on verified matches which we will use as true positives. We assess the level of agreement of the annotators using Fleiss Kappa coefficient for labeling the ground truth matchings in Table 4.3 and Table 4.4. The Kappa score we get is above 0.5, which is considered as a moderate agreement (0.41-0.60) [19].

D. The evaluation metrics. To compare our algorithms, we consider Precision, Relative Recall, and Relative F1-score for each algorithm in the experiment. The “Relative” term in the metrics, abbreviated Rel-, represents our effort to approximate the true Recall in the absence of established ground truth. Our goal is to *detect* an algorithm that will opt for high Precision at the cost of Recall in the context of the specific comparison. We approximate the number of real matches, which we do not know in our dataset, by providing a lower bound as follows. We take the union of all true positives (validated by our annotators) of all the algorithms in the test. Formally, we define I_{algo} to be the number of matches identified by algorithm *algo*. We define TP_{algo} to be the true positives for that algorithm as verified by the annotators. We then calculate the **union of the true positives** TP_{union} as the

Table 4.3: Performance comparison analysis for the algorithms in D_All using $SimT=0.7$ for GeekMAN with annotator agreement Kappa Score $K=0.53$.

Dataset	D_All			
Method	Precision	Rel-Recall	Rel-F1-score	K
Wang-16	76.0	29.9	42.9	0.53
UISN-UD	47.4	71.6	57.1	
GeekMAN	86.0	62.9	72.6	

Table 4.4: Performance comparison analysis for the algorithms in D_Slang using $SimT=0.7$ for GeekMAN with annotator agreement Kappa Score $K=0.54$.

Dataset	D_Slang			
Method	Precision	Rel-Recall	Rel-F1-score	K
Wang-16	77.7	25.3	38.1	0.54
UISN-UD	45.9	68.6	54.9	
GeekMAN	81.6	69.9	75.3	

union of all the true positives of all the algorithms: $TP_{union} = \bigcup_{a \in Algos} TP_a$. TP_{union} can be seen as a lower bound on the true matches that a perfect algorithm would have identified.

$$\text{Precision} = \frac{TP_{algo}}{I_{algo}} \quad \text{Rel-Recall} = \frac{TP_{algo}}{TP_{union}}$$

$$\text{Rel-F1-score} = \frac{2 * \text{Precision} * \text{Rel-Recall}}{\text{Precision} + \text{Rel-Recall}}$$

As the names indicate, the Relative Recall and Relative F1-score have only relative meaning within the scope of the comparison with the specific set of algorithms.

Table 4.5: Performance metrics between GeekMAN and the baseline algorithms across social network ground truth.

Dataset	D_Social		
Method	Precision	Recall	F1-Score
Wang-16	63.2	74.5	68.4
UISN-UD	66.1	75.1	70.3
GeekMAN ($SimT = 0.7$)	98.6	52.4	68.4

E. Choosing the Similarity Score Threshold. We want to identify an appropriate Similarity Score Threshold ($SimT$) value, which is a critical parameter for our approach. First, we apply GeekMAN and baseline algorithms on D_All (D_Multi + D_Slang) to find matching pairs between *source* forum and *target* forum. Second, the matchings are labeled by the annotators. Depending on their annotations, we calculate the defined evaluation metrics for our algorithm at different $SimT$ values ranging between 0.1-1.0 at 0.01 intervals. Finally, we plot the Precision, Rel-Recall, Rel-F1-score curves at those values in Figure 4.6. We find that the curves meet at $SimT$ value of 0.64. We notice that after $SimT = 0.68$, Rel-F1-score decreases, while Precision is increasing even after 0.70. We opt to prioritize Precision, and we choose a $SimT$ of 0.7 for our study.

F. Evaluation results. We show the performance evaluation for GeekMAN and the two baseline algorithms.

GeekMAN: 15% better Relative F1-score. In Table 4.3, we show the results for the D_All dataset. GeekMAN achieves a Precision of 86.0% and Relative F1-score of 72.6%. By contrast, the baseline algorithms achieve 76.0% and 47.4% Precision with 42.9% and 57.1%

Relative F1-score, respectively. It is also worth noting that the baseline algorithms only do well either in Precision or in Relative Recall. For example, Wang-16 offers high Precision (76.0%) but at the cost of the Rel-Recall (29.9%). The opposite is true for UISN-UD.

Observation 10. *GeekMAN outperforms both baselines on technogeek usernames with 86.0% Precision and 72.6% Rel-F1-score, while baseline algorithms achieve a maximum of 76.0% Precision or 57.1% Rel-F1-score, but never both.*

What about bona fide technogeek names? We want to further understand how the algorithms perform when the usernames use slang conventions. As expected, our approach does even better here with a difference in the Relative F1-score close to 20%. Focusing on our D_Slang dataset, we show the results in Table 4.4. GeekMAN surpasses the baselines in all metrics with a Precision of 81.6% and Relative F1-score of 75.3%, while baseline algorithms achieve a maximum 77.7% Precision and 54.9% Relative F1-score.

Observation 11. *On bona fide technogeek slang-style usernames, GeekMAN widens the gap to roughly 20% in Rel-F1-score over the best baseline.*

What do the other approaches miss? We did a deep dive to understand the origin of the lower performance of the other methods. Table 4.6 shows some of the matching pairs identified by GeekMAN that the baselines could not identify. We conjecture that this happens due to the usernames exhibiting slangification and chunkification, like *h4ckj4ck* and *JackTheHack*. These usernames exhibit slangification and chunkification, that the baselines were not designed to handle. For some of these algorithms as they rely on standard string techniques, such as 2-grams and substring matching, that will clearly not work well here.

Table 4.6: Real username matches that baseline algorithms missed.

Platform 1	Platform 2
c0d3m@st3r	Codomaster
41ph4w01f	alphawolffff
h4ckj4ck	JackTheHack
OverlordShadow	Shadowlord

G. GeekMAN performs well with “regular” usernames. We also show that our approach works well with regular social media usernames with Experiment-R. We apply each algorithm on D_Social to evaluate the performance. GeekMAN outperforms the baseline algorithms in terms of Precision being 98.6% with Similarity Score Threshold, $SimT = 0.7$, whereas the baseline algorithms have a maximum 66.1% Precision. Additionally, the F1-score of the methods is within 2% of each other.

4.4.2 Case study: Matching GitHub malware authors

We showcase the capabilities of our approach with the following application. We use GeekMAN to identify forum users that match malware authors on GitHub [85].

A. GeekMAN matches forum users with malware authors on GitHub. We find 6327 GitHub malware authors who have been matched with at least one user from one of our security forums. We apply our method with GitHub as the *source* platform and each of the security forums as the *target* forum. In the case that a study values Precision over Recall, we can adjust the Similarity Score Threshold. By using $SimT = 0.9$, we identify 1958 matches, which is an even stronger indication that a substantial set of these usernames may belong

Table 4.7: GeekMAN finds more forum-GitHub matches ($SimT = 0.7$) compared to exact matches alone. A substantial percentage of them are heavy-hitters in that forum. Note that a malware author can match users in more than one forum.

Forum	GitHub Author Matched		Heavy-hitters (%)
	Exact Match	GeekMAN	
GH	3	226	13.2
OC	16	1306	3.7
RF	20	2553	7.3
MP	64	4767	10.9
HF	182	5111	9.9

to the same user. We also find 260 GitHub malware authors in the security forums using exact username matching. For comparison, Table 4.7 shows the number of matched forum users using GeekMAN and exact username matching. We find that GeekMAN provides 24 and 7.5 times more matching than purely exact username matching using $SimT = 0.7$ and 0.9, respectively.

Observation 12. *As a use case on real cybercriminal data, GeekMAN identifies 6327 security-forum users that match malware authors on GitHub, of which only 260 are exact string matches.*

B. Deep dive: influential users and motives. Who are these malware authors and what is their activity on the security forums? Among the matched malware authors, we focus on the most influential authors who have at least 250 followers on GitHub. We then analyze the threads and posts of the counterpart matching forum users. They seem to fall

into the following broad groups according to major activity: (a) promoting their GitHub repositories (e.g. *TheSph1nx*, *TheSphinx*), (b) requesting hacking related information (e.g. *Cr4sh*, *crash*), (c) sharing hacking tutorials (e.g. *Bl4ckDr460n*, *blackdrag0*), and (d) boasting and advertising personal hacking skills (e.g. *Anon-Exploiter*, *An0n3xploit3r*). We also detect 4 matched GitHub authors who have since had their GitHub accounts deactivated, such as author *bl4ckic3* who was promoting a keylogger repository using the username *Blackice* in forum HF.

Many malware authors (3-13%) are highly active in the forums. We use the term **heavy-hitter** to refer to users in the top 10% by number of posts. In Table 4.7, we see that 3-13% of matched malware authors are also heavy-hitters in the forum, and 18 of them share their own GitHub profile and repositories in the URLs they post.

4.5 Discussion: Tool and Scope

We discuss the GeekMAN tool, potential scope and use case, the limitations and impact of our approach in this section.

The GeekMAN online platform. We provide an online platform that implements our approach to enable further research in the area. A sample of our platform is shown in Figure 4.7, which is publicly accessible at <https://geekman.streamlit.app>. Currently, we provide a set of minimal functionalities.

However, in our platform, we intend to support the following capabilities incrementally.

a. Finding a match within our database(s). A user can provide a username which we will match with the top-k most similar usernames in our database. A user can specify which platforms to include in the match, e.g. specific online forums, or all of them.

b. Leveraging our matching algorithms. A user can upload two CSV files with usernames, and we can provide the best matches for each username in the first file within the second file. The user can specify which algorithms to use for the matching, and even compare the similarity scores for the algorithms.

c. Providing the largest username database, ground truth and slang-dictionary. We intend to develop the largest database of usernames by adding more platforms that focus on technogeek space. We will also provide our manually verified results as ground truth, which we expect to grow over time with the help of experts and maybe even using carefully-run MTurk studies. Finally, we will share our technogeek dictionary which already has 5.8K entries.

Scope, practical applications and impact. We see our approach as a key building block for tracing malicious users across online platforms. Its focus on technogeek usernames makes it particularly well suited for the disambiguation of such users which, apart from malicious hackers, could include hate groups, and sociopaths, which could be of interest to law enforcement entities. GeekMAN amplifies the ability of security analysts to quickly hone in on *persons of interest* and comprehensively track them across platforms. We see it as the first step in the following pipeline: (a) collecting usernames from forums of interest, (b) identifying likely similar usernames, (c) investigating further with more computationally-

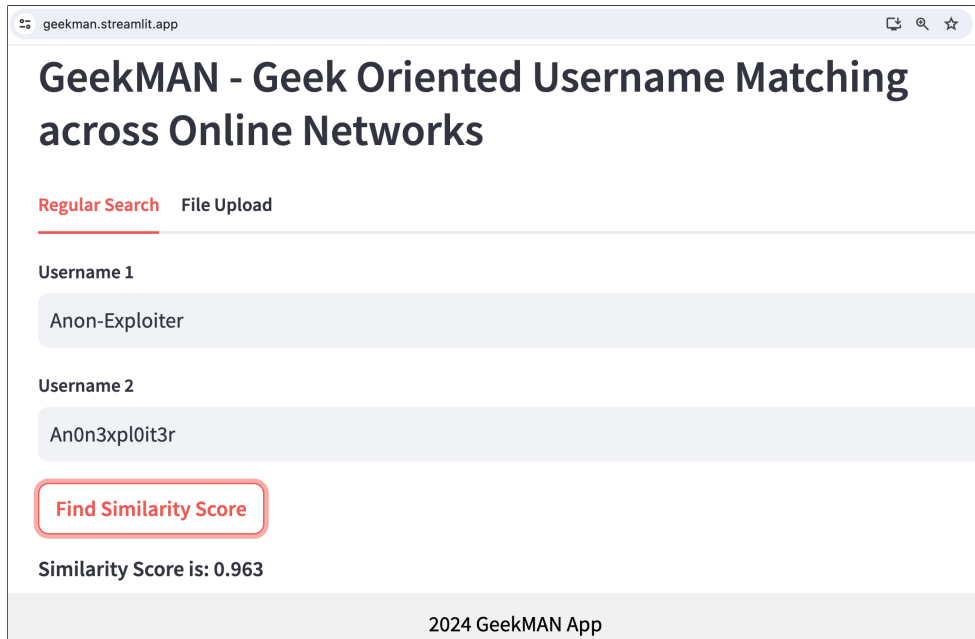


Figure 4.7: GeekMAN Platform: Matching usernames *Anon-Exploiter* and *An0n3xpl0it3r*. The platform is publicly accessible at <https://geekman.streamlit.app>.

expensive methods, and (d) involving humans in the effort. The goal is to enable efforts to stop individuals that cause real harm through cyber-criminal activities, such as ransomware, Denial of Service attacks, and identity theft. We would like to stress that a username match should be treated carefully as we discuss below.

How representative is our data? This is the question that plagues any measurement-driven effort. We argue that our data is sufficiently representative for the purposes of the study. The security forums that we use are among the appropriate targets that a security analyst would examine: they are highly technical and in the grey area between white and black hat hacking. Our successful linking of malware authors from GitHub to users in our forums is an additional indication of using relevant datasets. Our goal is to show the challenges and offer a solution in disambiguating technogeek usernames.

Limitations and extensions. We discuss limitations and potential extensions of the work.

Username vs. identity. It should be clear that identifying similar or even identical usernames does not guarantee that the owners behind them are the same person. However, we argue that this is a good place to start in absence of other leads. Our automatic sifting through hundreds of thousands or millions of usernames is essential to make such an investigation feasible.

Expanding the matching capability. We intend to further develop the matching capability by considering additional sources of information when these are available. For example, we will consider (a) the content of the posts, (b) posted or linked URLs, and (c) social connectivity. For the latter, we can extract “social links” by finding groups of users that post in the same threads, or that refer to each others’ resources and accounts on YouTube and GitHub.

Extensions. We intend to scale up our study by: (a) collecting data from more security forums and (b) interconnecting their users with other platforms, including YouTube, which provides hacking tutorials, communication platforms like Telegram and Discord, and social media, which are known to harbor hackers (from personal communication with Amazon’s security team).

Ethical considerations. Our work adheres to the community-defined ethical practices in particular the AAAI Code of Ethics and Professional Conduct [4]. First, we only use publicly available data. Anyone can collect the data from the forums and we use other data that has been made public by other researchers. Second, the usernames are not directly revealing any particular individual, but only represent an online persona. We only use

specific usernames to illustrate naming conventions but we will be happy to obfuscate or use artificial examples in the final version of the journal. Finally, we want to stress: a pair of matching usernames simply points to the probability that there is a connection and further proof should be solicited before any action is taken.

4.6 Related Work

Most previous works differ from our approach in that: (a) they are supervised approaches that need training data, (b) they are not focusing on complex-technogeek usernames, and (c) they rely on information beyond the username, such as user profile attributes, content, and social connectivity. By contrast, our approach is designed to: (a) handle technogeek names, and (b) rely only on usernames.

a. Username-based matching. We already discussed the two methods that we use in our performance analysis; [110] and [61].

Earlier, [81] employed a Markov-Chain based language model to quantify username uniqueness. The authors suggest that the selection of usernames follows a probabilistic distribution, with more predictable and less diverse usernames being more likely to be duplicated. To quantify this probability, they estimate the probability of a username’s occurrence based on a large username dataset leveraging Markov Chains model. To measure the similarity between usernames, they employ distance metrics such as Levenshtein distance and Jaro distance, alongside term frequency-inverse document frequency (TF-IDF).

Other efforts by [114] proposed MOBIUS to identify users across sites by analyzing the naming patterns of usernames. This technique exploits the redundant information in user-

name patterns revealed through user behavior. MOBIUS considers various user behavioral patterns, such as keyboard layouts, language usage, and username modification habits, to construct features. To link usernames, it employs several similarity measures, including Longest Common Substring (LCS), edit distance, Dynamic Time Warping distance, Jensen-Shannon divergence, and n-gram algorithm.

However, both previous approaches did not address the unique characteristics of usernames in the technogeek forum context as GeekMAN does. Additionally, technogeek users frequently engage as silent observers in the forum, leading to the unavailability of their behavioral patterns. This poses a challenge for earlier username disambiguation methods that rely on such data.

b. Using user profiles. An earlier study by [107] introduced a vector-based supervised algorithm which utilizes user profile features with differing weights. In some other efforts, [37] and [117] proposed probabilistic classifiers which link user identities based on profile attributes like description, location, and profile image. A projection-based modeling proposed by [74] incorporated profile features across platforms.

c. Combining multiple types of information. A number of efforts like [67, 121, 115, 120, 63] consider the social relationship for user matching. On the other hand, [62, 53, 36, 3, 8] incorporate a set of user related features including profile attributes, social relationships, and user generated contents. Some recent approaches [118, 116] leverage a user’s ego network to connect users across platforms.

4.7 Conclusion

We propose GeekMAN, a systematic approach to identify similar technogeek usernames across online platforms. The key novelty consists of three integrated capabilities: (a) decomposing usernames into meaningful chunks, (b) de-obfuscating technical and slang conventions, and (c) considering all the different outcomes of the two previous functions exhaustively when calculating the similarity. These three capabilities attempt to emulate the way a human will attempt to “understand” a username. We conduct a study using 1.8M usernames from three different types of forums: (a) security forums, (b) malware authors from GitHub, and (c) mainstream social media platforms. Our experiment shows that GeekMAN can match technogeek usernames with a precision of 81.6% and above, outperforming the previous studies. Our approach also outperforms prior methods on common social-network usernames in terms of precision. In the future, we intend to apply our method in other varied forums and platforms to interconnect users. Overall, we see our approach as a fundamental building block for tracing users across platforms, and we release (a) our platform, (b) datasets, (c) ground truth, and (d) a slang-translation dictionary.

Chapter 5

Conclusions

Our thesis proposes and develops a systematic suite of methods and tools to mine the socio-technical activities of an open-source platform. We develop robust tools to (a) identify repositories of interest at GitHub scale via query-by-example search, (b) study a community of repositories as a software ecosystem on GitHub, and (c) connect users of interest across platforms when their usernames are obfuscated and additional profile signals are unavailable. Our approaches have the following main advantages: (a) we develop complete capabilities for each task that can be operated even by non-experts; (b) our tools provide interpretable outputs (ranked repositories with cluster-level summaries, engineering archetypes with participation roles, and similarity scores with tunable thresholds); (c) our methods are designed to scale to public-platform sizes via offline crawls, embedding-based retrieval, and LLM-agent pipelines; and (d) we release datasets, ground truth, and online tools to maximize the reproducibility and reuse of this work.

Our study concludes in three key takeaway messages. First, the public surface of GitHub contains far more accessible information than its native search exposes, and a multi-granularity embedding-based capability can recover this information at platform scale: with RepoScope, users can find functionally similar repositories across hundreds of millions of public repositories with 86% top-10 precision and a 100% success rate, compared with the 66% precision that the GitHub API achieves on the same query set. Second, the malicious-intent slice of GitHub behaves as a wide-but-shallow software ecosystem: with MAGNET, we show that 80% of identified malicious-intent repositories are ephemeral one-time uploads, the sustained minority concentrates almost all observable engineering activity, and the surrounding active community of approximately 51K users across 711 repositories is dominated by following rather than co-development, a structure that any platform-governance or threat-intelligence intervention should account for. Third, hackers leave online footprints and consciously brand themselves across platforms, but their usernames are obfuscated in ways that defeat string matching: with GeekMAN, we show that a human-inspired pipeline that combines chunkification, deslangification, and exhaustive comparison can recover thousands of cross-platform matches at controllable precision, and we provide a slang-translation dictionary that can support follow-on studies. Our initial findings are just the beginning of a promising research direction that can shed light on the socio-technical ecosystem of public software-hosting platforms.

The current work thus can be seen as a building block that enables new research directions. Follow-up efforts can use our approach to (a) build continually updated query-by-example services for GitHub and adjacent platforms such as GitLab and BitBucket, (b) extend the

MAGNET framework to benign software-engineering ecosystems and to other code-hosting platforms where reputation and social interaction co-occur, (c) integrate GeekMAN with content-level signals (topical similarity, social ties, and stylometric features) to push cross-platform user tracing further, and (d) study the interplay between repository influence, author reputation, and downstream platform behavior using the unified data products that this thesis releases. We believe that a continued, principled, end-to-end mining of public software-hosting platforms is essential both for advancing software-engineering research and for safeguarding the open-source ecosystem.

Bibliography

- [1] Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. Can llms replace manual annotation of software engineering artifacts? 2025.
- [2] GitHub API. Github api. <https://api.github.com>, 2024. Accessed: 2024-11-08.
- [3] Ehsan Arabnezhad, Massimo La Morgia, Alessandro Mei, Eugenio Nerio Nemmi, and Julinda Stefa. A light in the dark web: Linking dark web aliases to real internet identities. In *ICDCS*, 2020.
- [4] Association for the Advancement of Artificial Intelligence. AAAI code of ethics and professional conduct. <https://aaai.org/about-aaai/ethics-and-plurality/>, 2025. Accessed: 2026-05-28.
- [5] Muhammad Hilmi Asyrofi, Ferdian Thung, David Lo, and Lingxiao Jiang. Ausearch: Accurate api usage search in github repositories with type resolution. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 637–641. IEEE, 2020.
- [6] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. Llm2vec: Large language models are secretly powerful text encoders. *arXiv preprint arXiv:2404.05961*, 2024.
- [7] Mikhail Bilenko, Raymond Mooney, William Cohen, Pradeep Ravikumar, and Stephen Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 2003.
- [8] José Cabrero-Holgueras and Sergio Pastrana. A methodology for large-scale identification of related accounts in underground forums. *Computers & Security*, 2021.
- [9] Alejandro Calleja, Juan Tapiador, and Juan Caballero. A look into 30 years of malware development from a software metrics perspective. In *Research in Attacks, Intrusions, and Defenses*, pages 325–345, 2016.
- [10] Alejandro Calleja, Juan Tapiador, and Juan Caballero. The malsource dataset: Quantifying complexity and code reuse in malware development. *IEEE Transactions on Information Forensics and Security*, 14(12):3175–3190, 2018.

- [11] Cambridge Cybercrime Centre. CrimeBB Dataset: Cambridge Cybercrime Centre Data Sharing Programme. <https://www.cambridgecybercrime.uk>, 2022. Accessed: 2025-12-15.
- [12] Junkai Chen, Xing Hu, Zhenhao Li, Cuiyun Gao, Xin Xia, and David Lo. Code search is all you need? improving code suggestions with code search. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.
- [13] Minyu Chen, Guoqiang Li, Chen Ma, Jingyang Li, and Hongfei Fu. Repo4qa: Answering coding questions via dense retrieval on github repositories. In *29th International Conference on Computational Linguistics (COLING 2022)*, pages 1580–1592, 2022.
- [14] Valerio Cosentino, Javier Luis, and Jordi Cabot. Findings from github: methods, datasets and limitations. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 137–141, 2016.
- [15] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in github for msr studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 560–564. IEEE, 2021.
- [16] Edwin S Dalmajer, Camilla L Nord, and Duncan E Astle. Statistical power for cluster analysis. *BMC bioinformatics*, 23(1):205, 2022.
- [17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [18] Roberto Di Cosmo and Stefano Zacchiroli. Software heritage: Why and how to preserve software source code. In *iPRES 2017-14th International Conference on Digital Preservation*, pages 1–10, 2017.
- [19] Khaled El Emam. Benchmarking kappa: Interrater agreement in software process assessments. *Empirical Software Engineering*, 1999.
- [20] facebookresearch. facebookresearch. <https://github.com/facebookresearch/faiss>, 2025. Accessed: 2025-03-05.
- [21] Federal Bureau of Investigation. Behzad Mohammadzadeh — FBI Cyber’s Most Wanted. <https://www.fbi.gov/wanted/cyber/behzad-mohammadzadeh>, 2020. Accessed: 2025-12-15.
- [22] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.
- [23] Garage4Hackers. Garage4Hackers: An Information Security Community. <http://garage4hackers.com>, 2021. Accessed: 2025-12-15.

- [24] GhArchive. Gharchive. <https://www.gharchive.org/>. [Online; accessed 13-Feb-2024].
- [25] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. Some from here, some from there: Cross-project code reuse in github. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 291–301. IEEE, 2017.
- [26] Joobin Gharibshah, Evangelos E Papalexakis, and Michalis Faloutsos. RIPEX: Extracting malicious ip addresses from security forums using cross-forum learning. In *PAKDD*, 2018.
- [27] Joobin Gharibshah, Evangelos E Papalexakis, and Michalis Faloutsos. REST: A thread embedding approach for identifying and classifying user-specified information in security forums. In *ICWSM*, 2020.
- [28] Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. Chatgpt outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences*, 120(30):e2305016120, 2023.
- [29] GitHub. Github. <https://github.com/about>, 2024. Accessed: 2024-11-08.
- [30] GitHub. Github graphql api. <https://docs.github.com/en/graphql>, 2024. Accessed: 2024-11-08.
- [31] GitHub. Github. <https://github.com/code4craft/webmagic>, 2025. Accessed: 2025-03-05.
- [32] GitHub. Github. <https://github.com/nehamchangappa/Online-Banking-Web-Applcation-Project>, 2025. Accessed: 2025-03-05.
- [33] GitHub. Github. <https://github.com/davidson679/Bypass-Web-Application-Firewalls>, 2025. Accessed: 2025-03-05.
- [34] GitHub. Githubapi. <https://api.github.com/repositories?since=1>, 2025. Accessed: 2025-03-05.
- [35] GitHub. Octoverse 2025: A new developer joins GitHub every second as AI leads TypeScript to #1. <https://github.blog/news-insights/octoverse/octoverse-a-new-developer-joins-github-every-second-as-ai-leads-typescript-to-1/>, 2025. Accessed: 2025-12-15.
- [36] Oana Goga, Patrick Loiseau, Robin Sommer, Renata Teixeira, and Krishna P Gumadi. On the reliability of profile matching across large online social networks. In *ACM SIGKDD*, 2015.
- [37] Oana Goga, Daniele Perito, Howard Lei, Renata Teixeira, and Robin Sommer. Large-scale correlation of accounts across social networks. *University of California at Berkeley, Berkeley, California, Tech. Rep. TR-13-002*, 2013.

- [38] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on Github. In *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020.
- [39] Georgios Gousios. The ghtorrent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 233–236. IEEE, 2013.
- [40] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.
- [41] Kilem L. Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, 2008.
- [42] Hack Forums. Hack Forums: Hacking and Information Security Community. <https://hackforums.net>, 2021. Accessed: 2025-12-15.
- [43] Patrick AV Hall and Geoff R Dowling. Approximate string matching. *ACM computing surveys (CSUR)*, 1980.
- [44] Foyzul Hassan, Shaikh Mostafa, Edmund SL Lam, and Xiaoyin Wang. Automatic building of java projects in software repositories: A study on feasibility and challenges. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 38–47. IEEE, 2017.
- [45] Lars Heinemann, Florian Deissenboeck, Mario Gleirscher, Benjamin Hummel, and Maximilian Irlbeck. On the extent and nature of software reuse in open source java projects. In *Top Productivity through Software Reuse: 12th International Conference on Software Reuse, ICSR 2011, Pohang, South Korea, June 13-17, 2011. Proceedings 12*, pages 207–222. Springer, 2011.
- [46] hoffa. medium. <https://hoffa.medium.com/400-000-github-repositories-1-billion-files-14-terabytes-of-code-spaces-or-tabs-7cfe0b5dd7fd>, 2016. Accessed: 2025-03-03.
- [47] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 2023.
- [48] Yan Hu, Jun Zhang, Xiaomei Bai, Shuo Yu, and Zhuo Yang. Influence analysis of Github repositories. *SpringerPlus*, 5(1):1–19, 2016.
- [49] Risul Islam, Md Omar Faruk Rokon, Ahmad Darki, and Michalis Faloutsos. Hackerscope: The dynamics of a massive hacker online ecosystem. *Social Network Analysis and Mining*, 11(1):1–12, 2021.

- [50] Risul Islam, Md Omar Faruk Rokon, Evangelos E Papalexakis, and Michalis Faloutsos. Recten: A recursive hierarchical low rank tensor factorization method to discover hierarchical patterns from multi-modal data. In *ICWSM*, 2021.
- [51] Risul Islam, Md Omar Faruk Rokon, Evangelos E Papalexakis, and Michalis Faloutsos. Tenfor: Tool to mine interesting events from security forums leveraging tensor decomposition. In *Social Media Analysis for Event Detection*, pages 57–87. Springer, 2022.
- [52] Mahmoud Jahanshahi, David Reid, and Audris Mockus. Beyond dependencies: The role of copy-based reuse in open source software development. *ACM Transactions on Software Engineering and Methodology*, 2024.
- [53] Paridhi Jain, Ponnurangam Kumaraguru, and Anupam Joshi. @ i seek’fb. me’ identifying users across multiple online social networks. In *WWW*, 2013.
- [54] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Deendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- [55] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21:2035–2071, 2016.
- [56] Shinji Kawaguchi, Pankaj K Garg, Makoto Matsushita, and Katsuro Inoue. Mudblue: An automatic categorization system for open source repositories. *Journal of Systems and Software*, 79(7):939–953, 2006.
- [57] Sangeeth Kochanthara, Yanja Dajsuren, Loek Cleophas, and Mark van den Brand. Painting the landscape of automotive software in github. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 215–226, 2022.
- [58] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, 1966.
- [59] Agathe Lherondelle, Varun Babbar, Yash Satsangi, Fran Silavong, Shaltiel Eloul, and Sean Moran. Topical: Automatic repository tagging using attention on hybrid code embeddings. In *Proceedings of the 1st IEEE/ACM Workshop on Software Engineering Challenges in Financial Firms*, pages 23–30, 2024.
- [60] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.
- [61] Yongjun Li, You Peng, Zhen Zhang, Hongzhi Yin, and Quanqing Xu. Matching user accounts across social networks based on username and display name. *World Wide Web*, 2019.

- [62] Jing Liu, Fan Zhang, Xinying Song, Young-In Song, Chin-Yew Lin, and Hsiao-Wuen Hon. What’s in a name? an unsupervised approach to link users across communities. In *WSDM*, 2013.
- [63] Li Liu, William K Cheung, Xin Li, and Lejian Liao. Aligning users across social networks using network embedding. In *IJCAI*, 2016.
- [64] Yuyang Liu, Ehsan Noei, and Kelly Lyons. How readme files are structured in open source java projects. *Information and Software Technology*, 148:106924, 2022.
- [65] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretski, and Audris Mockus. World of code: an infrastructure for mining the universe of open source vcs data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 143–154. IEEE, 2019.
- [66] Yuxing Ma, Tapajit Dey, Chris Bogart, Sadika Amreen, Marat Valiev, Adam Tutko, David Kennard, Russell Zaretski, and Audris Mockus. World of code: enabling a research workflow for mining and analyzing the universe of open source vcs data. *Empirical Software Engineering*, 26:1–42, 2021.
- [67] Anshu Malhotra, Luam Totti, Wagner Meira Jr, Ponnurangam Kumaraguru, and Virgilio Almeida. Studying user footprints in different online social networks. In *ASONAM*, 2012.
- [68] Md Rayhanul Masud. GeekMAN: Geek-oriented Username Matching Across Online Networks — Source Code and Datasets. <https://github.com/mrayhanulmasud/geekman>, 2023. Accessed: 2025-12-15.
- [69] Md Rayhanul Masud, Ben Treves, and Michalis Faloutsos. Geekman: Geek-oriented username matching across online networks. In *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*, pages 305–309, 2023.
- [70] Collin McMillan, Mark Grechanik, and Denys Poshyvanyk. Detecting similar software applications. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 364–374. IEEE, 2012.
- [71] Microsoft Threat Intelligence. Malvertising campaign leads to info stealers hosted on github. <https://www.microsoft.com/en-us/security/blog/2025/03/06/malvertising-campaign-leads-to-info-stealers-hosted-on-github/>, Mar 2025. Accessed: 2026-05-18.
- [72] Alvaro E Monge, Charles Elkan, et al. The field matching problem: algorithms and applications. In *KDD*, 1996.
- [73] MPGH. MultiPlayer Game Hacking (MPGH): Game Hacking and Cheating Community. <https://www.mpgh.net>, 2021. Accessed: 2025-12-15.
- [74] Xin Mu, Feida Zhu, Ee-Peng Lim, Jing Xiao, Jianzong Wang, and Zhi-Hua Zhou. User identity linkage by latent user space modelling. In *ACM SIGKDD*, 2016.

- [75] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.
- [76] Kawser Wazed Nafi, Banani Roy, Chanchal K Roy, and Kevin A Schneider. A universal cross language software similarity detector for open source software categorization. *Journal of Systems and Software*, 162:110491, 2020.
- [77] Phuong T Nguyen, Juri Di Rocco, Riccardo Rubei, and Davide Di Ruscio. Crosssim: exploiting mutual relationships to detect similar oss projects. In *2018 44th Euromicro conference on software engineering and advanced applications (SEAA)*, pages 388–395. IEEE, 2018.
- [78] Phuong T Nguyen, Juri Di Rocco, Riccardo Rubei, and Davide Di Ruscio. An automated approach to assess the similarity of github repositories. *Software Quality Journal*, pages 1–37, 2020.
- [79] Offensive Community. Offensive Community: Underground Hacking and Security Forum. <http://offensivecommunity.net>, 2021. Accessed: 2025-12-15.
- [80] Sergio Pastrana, Daniel R. Thomas, Alice Hutchings, and Richard Clayton. Crimebb: Enabling cybercrime research on underground forums at scale. In *Proceedings of The Web Conference 2018*, pages 1845–1854, 2018.
- [81] Daniele Perito, Claude Castelluccia, Mohamed Ali Kaafar, and Pere Manils. How unique and traceable are usernames? In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2011.
- [82] Yiyue Qian, Yiming Zhang, Qianlong Wen, Yanfang Ye, and Chuxu Zhang. Rep2Vec: Repository embedding via heterogeneous graph adversarial contrastive learning. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.
- [83] RaidForums. RaidForums: Database Sharing and Hacking Discussion Forum. <https://raidforums.com>, 2021. Seized by the U.S. Department of Justice in 2022; archived snapshots used in this study. Accessed: 2025-12-15.
- [84] A Eashaan Rao and Sridhar Chimalakonda. Apples, oranges & fruits—understanding similarity of software repositories through the lens of dissimilar artifacts. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 384–388. IEEE, 2022.
- [85] Md Omar Faruk Rokon, Risul Islam, Ahmad Darki, Evangelos E Papalexakis, and Michalis 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)*, pages 149–163, 2020.
- [86] Md Omar Faruk Rokon, Pei Yan, Risul Islam, and Michalis Faloutsos. Repo2vec: A comprehensive embedding approach for determining repository similarity. In *2021*

- IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 355–365. IEEE, 2021.
- [87] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.
 - [88] Pamela H Russell, Rachel L Johnson, Shreyas Ananthan, Benjamin Harnke, and Nichole E Carlson. A large-scale analysis of bioinformatics code on github. *PloS one*, 13(10):e0205898, 2018.
 - [89] Nareerat Saetiew, Tiranee Achalakul, and Santitham Prom-on. Thai person name recognition (pnr) using likelihood probability of tokenized words. In *2017 International Electrical Engineering Congress (iEECON)*, pages 1–4. IEEE, 2017.
 - [90] Sagar Samtani and Hsinchun Chen. Using social network analysis to identify key hackers for keylogging tools in hacker forums. In *Intelligence and Security Informatics (ISI)*. IEEE, 2016.
 - [91] Anna Sapienza, Alessandro Bessi, Saranya Damodaran, Paulo Shakarian, Kristina Lerman, and Emilio Ferrara. Early warnings of cyber threats in online discussions. In *2017 IEEE International Conference on Data Mining Workshops*, pages 667–674, 2017.
 - [92] Ketan Rajshekhar Shahapure and Charles Nicholas. Cluster quality analysis using silhouette score. In *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*, pages 747–748. IEEE, 2020.
 - [93] Huajie Shao, Dachun Sun, Jiahao Wu, Zecheng Zhang, Aston Zhang, Shuochao Yao, Shengzhong Liu, Tianshi Wang, Chao Zhang, and Tarek Abdelzaher. paper2repo: Github repository recommendation for academic papers. In *Proceedings of The Web Conference 2020*, pages 629–639, 2020.
 - [94] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 314–319, 2017.
 - [95] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.
 - [96] SourceFinder. Finding malware source-code from publicly available repositories in github. <http://hackerchatter.org/sourcefinder/>. [Online; accessed 12-January-2021].
 - [97] Nishat Ara Tania, Md Rayhanul Masud, Md Omar Faruk Rokon, Qian Zhang, and Michalis Faloutsos. Who is creating malware repositories on github and why? In *Companion Proceedings of the ACM Web Conference 2024*, pages 955–958, 2024.

- [98] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.
- [99] Antonis Terefos. Stargazers ghost network. Technical report, Check Point Research, Jul 2024.
- [100] Michal Tereszkowski-Kaminski, Santanu Kumar Dash, and Guillermo Suarez-Tangil. A study of malicious source code reuse among github, stackoverflow and underground forums. In *European Symposium on Research in Computer Security*, pages 45–66. Springer, 2024.
- [101] Ferdian Thung, David Lo, and Lingxiao Jiang. Detecting similar applications with collaborative tagging. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 600–603. IEEE, 2012.
- [102] Ferdian Thung, David Lo, and Julia Lawall. Automated library recommendation. In *2013 20th Working conference on reverse engineering (WCRE)*, pages 182–191. IEEE, 2013.
- [103] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 163–166. IEEE, 2009.
- [104] Nguyen Huy Tien, Nguyen Minh Le, Yamasaki Tomohiro, and Izuha Tatsuya. Sentence modeling via multiple word embeddings and multi-level comparison for semantic textual similarity. *Information Processing & Management*, 56(6):102090, 2019.
- [105] V. A. Traag, L. Waltman, and N. J. van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019.
- [106] Oleg Vasilyev, Vedant Dharnidharka, and John Bohannon. Fill in the blanc: Human-free quality estimation of document summaries. *arXiv preprint arXiv:2002.09836*, 2020.
- [107] Jan Vosecky, Dan Hong, and Vincent Y Shen. User identification across multiple social networks. In *First International Conference on Networked Digital Technologies*. IEEE, 2009.
- [108] TANG Wan, HU Jun, Hui Zhang, WU Pan, and HE Hua. Kappa coefficient: a popular measure of rater agreement. *Shanghai archives of psychiatry*, 27(1):62, 2015.
- [109] Xinchun Wang, Ruida Hu, Cuiyun Gao, Xin-Cheng Wen, Yujia Chen, and Qing Liao. Reposvul: A repository-level high-quality vulnerability dataset. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pages 472–483, 2024.

- [110] Yubin Wang, Tingwen Liu, Qingfeng Tan, Jinqiao Shi, and Li Guo. Identifying users across different sites using usernames. *Procedia Computer Science*, 2016.
- [111] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.
- [112] Wikipedia contributors. Leet — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/wiki/Leet>, 2023. Accessed: 2025-12-15.
- [113] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.
- [114] Reza Zafarani and Huan Liu. Connecting users across social media sites: a behavioral-modeling approach. In *ACM KDD*, 2013.
- [115] Reza Zafarani, Lei Tang, and Huan Liu. User identification across social media. *ACM TKDD*, 2015.
- [116] Fanjin Zhang and Liu et al. Oag: Toward linking large-scale heterogeneous entity graphs. In *ACM SIGKDD*, 2019.
- [117] Haochen Zhang, Min-Yen Kan, Yiqun Liu, and Shaoping Ma. Online social network profile linkage. In *Asia Information Retrieval Symposium*. Springer, 2014.
- [118] Jing Zhang, Bo Chen, Xianming Wang, Hong Chen, Cuiping Li, Fengmei Jin, Guojie Song, and Yutao Zhang. Mego2vec: Embedding matched ego networks for user alignment across social networks. In *ACM CIKM*, 2018.
- [119] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. Detecting similar repositories on github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 13–23. IEEE, 2017.
- [120] Yutao Zhang, Jie Tang, Zhilin Yang, Jian Pei, and Philip S Yu. COSNET: Connecting heterogeneous social networks with local and global consistency. In *ACM SIGKDD*, 2015.
- [121] Xiaoping Zhou, Xun Liang, Haiyan Zhang, and Yuefeng Ma. Cross-platform identification of anonymous identical users in multiple social media networks. *IEEE transactions on knowledge and data engineering*, 2015.