MetaSim: A search engine for finding Similar GitHub Repositories

Md Rayhanul Masud
UC Riverside
mmasu012@ucr.edu

Md Omar Faruk Rokon
UC Riverside
mroko001@ucr.edu

Qian Zhang
UC Riverside
qzhang@cs.ucr.edu

Michalis Faloutsos
UC Riverside
michalis@cs.ucr.edu

***Abstract*—How can we find other repositories on GitHub that are functionally similar to a specific repository? While GitHub offers keyword-based search functionality, there is a lack of a tool that can perform query by example to search and compare functionally similar repositories. To address this challenge, we present MetaSim: a search engine that finds similar GitHub repositories based on repository metadata features. MetaSim employs a customized technique to represent repository metadata in the embedding space for efficient indexing and searching. We construct a curated dataset of 267.6K public GitHub repositories to support our search engine. We evaluate our tool through a manual assessment on a set of 202 query by example repository and their corresponding matching pairs. Experiment results demonstrate that *Readme* alone can achieve high similarity precision (90.1%), which we define later. In contrast, the combined usage of *Description*, *Topics*, and *Readme* yields the best overall performance with similarity precision of 97.8%. To foster both research and practical applications, we open source our research artifacts through the MetaSim platform at https://metasim-app.github.io. The demonstration video of MetaSim is available at https://youtu.be/HnFnN3JclQw.**

I. Introduction

The rise of open-source development has led to a significant increase in the use of GitHub repositories for learning and sharing knowledge [1]. Developers frequently explore GitHub to find similar repositories for software prototyping, explore alternative implementations, and contribute to relevant projects[2], [3]. Additionally, researchers collect GitHub repositories related to their specific research interests for software repository mining studies[4]. However, in the current era of automated software engineering driven by AI, the ability to efficiently identify repositories of interest has become increasingly crucial. This highlights the pressing need for a tool that can effectively identify functionally similar repositories.

Problem. This work is driven by the question: *Given a GitHub repository, how can we find the 50 most similar repositories in terms of functionality?* Establishing similarity of GitHub repositories is a challenging task for two reasons. First, the similarity can be defined in diverse ways based on the query's objectives. For example, we can focus on code level similarity [5] or put more weight on the high level functionality such as finding alternative algorithms [6], which necessitates a consideration of metadata information. Second, the format of the query introduces significant variations, with some queries rooted in examples, where users provide a target repository, while others could offer a set of keywords. This work focuses on the example-based query scenario focusing on the high-level capabilities of a repository which we refer to the term *functional similarity*. Finding such repositories necessitates the use of various metadata features, for which we use the most commonly used metadata features discussed in [7], [8] including *RepoName*, *Description*, *Topics*, and *Readme*.

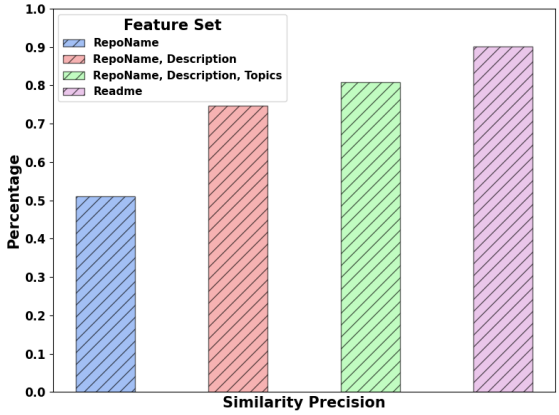

Fig. 1: Assessing the contribution of each metadata feature in determining repository similarity in terms of Similarity Precision.

Related work. Prior work in mining GitHub repositories infers the similarity by considering a combination of dimensions, including metadata, source code, and social context. In contrast, we aim to develop a tool that can support the search for finding similar repositories on GitHub.

Contribution. We present MetaSim, a search engine to identify similar GitHub repositories based on repository metadata features. MetaSim relies on two functionalities: (a) the *Meta-Representation*, which uses an embedding approach to represent a repository, (b) the *Meta-Similarity*, which measures the similarity between repositories. Specifically, our tool performs the following task: given the query repository and a set of target repositories, it ranks the target ones in order of similarity. To complement our search engine, we construct a curated dataset of 267.6K public GitHub repositories. We evaluate MetaSim through a manually annotation on a set of 202 query by-example (repository, matching) pairs. We define two evaluation metrics: (a) Similarity Precision, and (b) Success Rate to compare the performance of different combinations of features in finding similar repositories, as discussed in Section IV. We summarize the key results below.

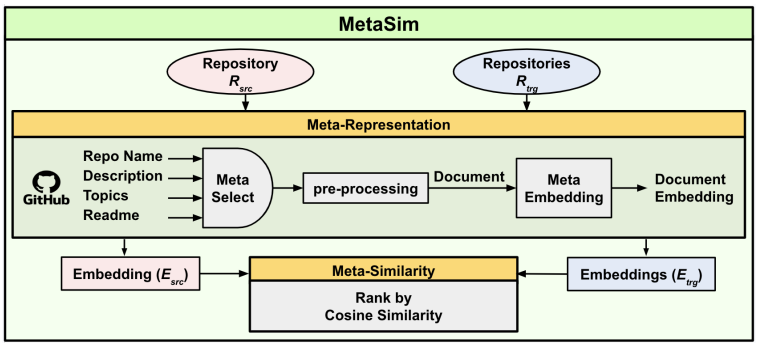

Fig. 2: MetaSim architecture diagram shows how Repository R_{src} is compared against a set of Target repositories R_{trg}.

A. *Readme* **is the most informative feature**. We find that using the *Readme* alone produces better search results than using the other three features together (*RepoName*, *Description*, *Topics*). Specifically, using only *Readme* results in a Similarity Precision of 90.1%, compared to 80.8% achieved with the three other features combined as shown in Figure 1.

B. *Description*, *Topics*, **and** *Readme* **together as a feature set stands the best performing.** This feature set together achieves 97.8% Similarity Precision and 100.0% Success Rate.

II. Dataset Overview

According to the study [9], there is a substantial amount of personal and duplicate projects in GitHub. So, different selection criteria are considered for collecting repositories for software engineering tasks. To collect repositories to create the underlying repository database for our MetaSim search engine, we select the following criteria: repository (a) is of size greater than 0KB, (b) has at least 5 stars and 5 forks, (c) is public and not forked from other repository. These criteria are adapted from [10], [11] to ensure that we can discard unused, duplicate and empty repositories. Since GitHub API returns maximum 1K results/query, we invoke the repository search API using the above criteria for each day in range from Jan'08 to Sep'23. As an example, the API [1] is invoked to retrieve repositories that are published on Jan 01, 2022. This results in 1.46M public GitHub repositories with metadata features: repo name, description, topics, stargazers, forks count and so on. Then we download readme content (if available) for those repositories. Then, we exclude repositories where any of our metadata features is empty or the content is in a non-English language. This refining process yields D_All, comprising 267.6K repositories.

III. Tool: MetaSim

To support the search of similar repositories, we have implemented a search engine MetaSim that consists of two functionalities: (A) Meta-Representation, and (B) Meta-Similarity, inspired by an earlier work Repo2Vec[7]. While the first function represents a repository as a document embedding combining a set of selected metadata features, the latter computes the similarity of a query repository R_{src} with the repositories from the database D_All, which we call as target repositories R_{trg}. All target repositories are represented in the embedding space and stored in a vector database. Then given a query repository from the user, similar repositories are returned based on the proximity of their embeddings with the query repository embedding. A conceptual overview of inner working of MetaSim is illustrated in Figure 2, and the web interface of our tool is displayed in Figure 3.

A. The Meta-Representation Function

In this step, a repository is represented in the vector space utilizing three sub-functionalities: (i) Meta Select, (ii) Pre-processing, and (iii) Meta Embedding. We discuss them in detail in the following.

Meta Select. This module selects different combinations of features to represent a repository. It is obvious that adding more metadata information can help create better representation, but it can come at the cost of computation. The module considers the following combinations of features: (i) single feature, (ii) two features, (iii) three features, (iv) all features from RepoName, Description, Topics, and Readme, and represent the repositories separately, each for the selected combination.

Meta Select selects combination of features, and feeds them to the next step for pre-processing.

Pre-processing. The features are pre-processed to discard unnecessary details, and concatenated together to form a document.

RepoName, Topics: *RepoName* is tokenized using the available delimiters ('-', '.', '_') and Camel Case conventions as followed in [12]. Then a single sentence is created by concatenating the tokens together. On the other hand, all of the *Topics* in a repository are also concatenated by a comma (,), and form another single sentence.

Description, Readme: *Description* of a repository sometimes contains URLs. In addition to URLs, *Readme*s often share source code, and provide contents embedded in html tags. The URLs, source code content, and html tags from *Description* and *Readme* are discarded.

[1] https://api.github.com/search/repositories?q=is:public+fork:false+stars:≥5+forks:≥5+size:>0+created:2022-01-01..2022-01-01

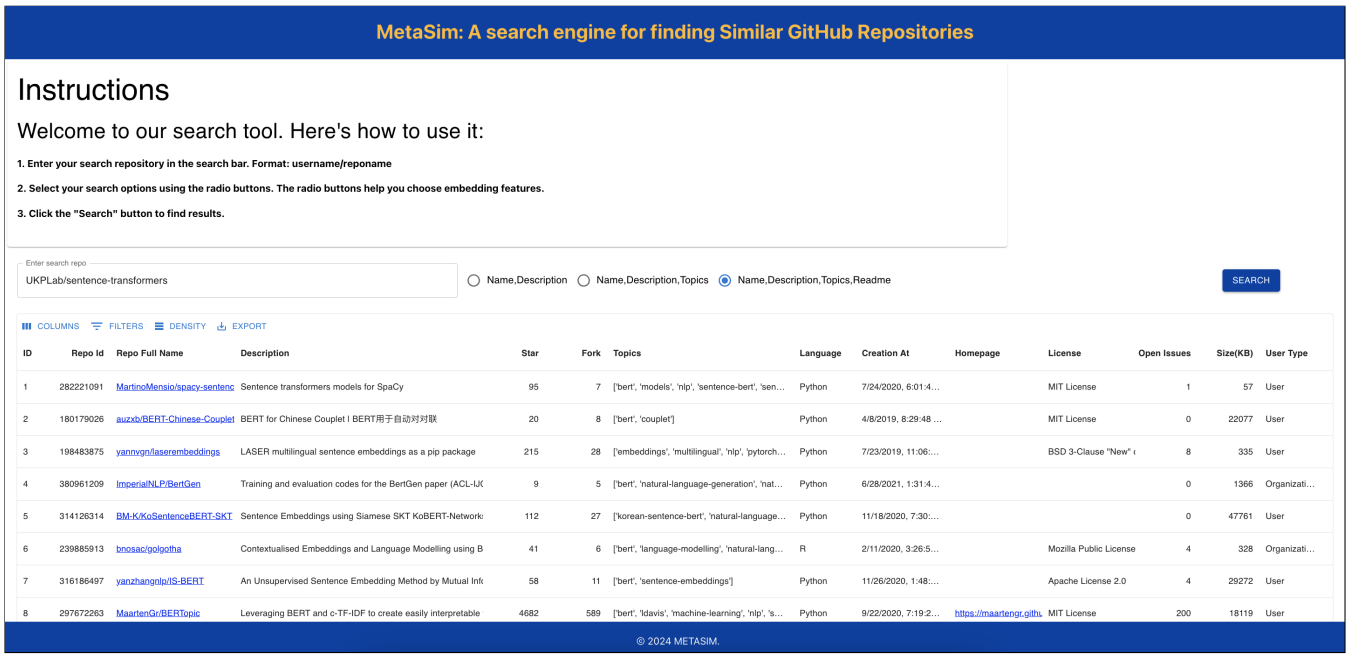

Fig. 3: MetaSim Tool: A search engine to find similar repositories.

Finally, the pre-processed selected features are joined together to form a representative text document from a given repository.

Meta Embedding. To represent a textual document in the vector space, we can choose different techniques that are varied in use-case and requirements. As our aim is to capture repository similarities based on their metadata, we estimate the semantic proximity across the metadata information. We follow the embedding approach from [13] that uses SentenceBERT [14] to generate embedding from a set of metadata features. Finally, for each of the feature combination, an embedding for a given repository is created from its textual representation.

B. The Meta-Similarity Function

We estimate the similarity of a source query repository R_{src} with the target repositories R_{trg} of D_All based on the similarities across their meta embedding computed in previous step. We use FAISS [2] (Facebook AI Similarity Search), an optimized and efficient similarity search library to compute the similarity. Using FAISS, we index and store list of meta embedding E_{trg} from all of the target repositories R_{trg} in D_All database, for each of the feature combinations discussed earlier. Finally, given a source repository embedding E_{src} and a feature set, this module ranks the repositories of D_All that are closest in embedding space for the given feature set in respect of cosine similarity.

C. User Interface

We provide an online platform for users to find similar GitHub repositories based on a given query repository. The web interface of the platform is displayed in Figure 3. We support the following functionalities.

1) A user can provide *author* and *RepoName* as query with a feature set of her interest to be used for finding similar repositories in D_All. The similar repositories are returned with their corresponding metadata features.
2) The user can filter the search results based on topics, programming language and license.
3) The user can order the search results by star/fork/open issue count and publish date.
4) The user can download the results in csv format.

IV. Experiment and Results

In this section, we discuss the ground-truth creation, experimental setup, and performance evaluation of different combinations of metadata feature to determine repository similarity.

A. Ground-truth Creation

To create the ground-truth search results, we first select a set of query repositories *QuerySet*. Next, we find similar repositories for *QuerySet* repositories considering all of our metadata features. These matches are manually validated by the annotators. This process yields *DGround*, which contains the union of all repositories in *QuerySet* along with their validated counterparts. Below we discuss the steps in detail.

***QuerySet* Selection**. We select *QuerySet* repositories to ensure that they can represent diverse topics and varying levels of popularity.

- *Identifying 30 Popular Topics:* We find that there are 193.3K unique topics in D_All. We calculate the frequency distribution of the topics. The distribution lets us group the most frequent topics in three categories:

[2] https://github.com/facebookresearch/faiss

TABLE I: Popular topics for creating ground-truth dataset, *DGround*

Topics		
Most Popular	**Medium Popular**	**Less Popular**
hacktoberfest, deep-learning, machine-learning, security, blockchain, computer-vision, open-source, data-science, visualization, database	rest-api, website, serverless, image-processing, natural-language-processing, reverse-engineering, game-development, cryptography, object-detection, hacking	algorithms, logging, pentesting, networking, privacy, education, microservices, ecommerce, analytics, design

(a) Most, (b) Medium, and (c) Less Popular. Then, we manually select 10 topics from each of these categories excluding the generic topics related to programming language or tools (e.g. python, java, vscode, linux) as shown in Table I.

- *Selecting QuerySet Repositories for Popular Topics:* To balance the size of *QuerySet* and the subsequent annotation cost, we opt for $n = 9$ repositories per popular topic. This results in a curated set of 270 repositories for *QuerySet*.

Repository Search. We find repositories similar to each of *QuerySet* repositories using MetaSim tool. We deploy each of *QuerySet* as R_{src} and all repositories of D_All as R_{trg}. We take five top matches per query repository. Here, Meta Select chooses the combination of all features (*RepoName*, *Description*, *Topics*, and *Readme*) to best approximate the ground-truth matchings.

Manual Validation. We allot 270 sets of records (record: query repository, 5 top matches) to 9 domain-expert computer scientists, where each of them is assigned 30 different query repositories and their corresponding matches. They assign a confidence score from 1 to 5 to each matching pair (with 5 corresponding to 'highly similar', 4 to 'similar', 3 to 'neutral', 2 to 'dissimilar', 1 to 'highly dissimilar'). The annotators have at least 2 years of experience of working in software industry. They are instructed to visit the repositories online, and assign the score considering all the repository information available including source code, metadata features and so on.

***DGround* Creation**. Based on the annotation, we find 68 query repositories having no similar/highly similar repository, and we discard them. Finally we take the union of *QuerySet* repositories and their corresponding manually annotated similar repos to create our ground-truth dataset, including 1199 repositories.

B. Experimental Setup and Evaluation Metrics

We run experiment and evaluate the performance of different combinations of feature set to find repository similarity.

In particular, for each feature set, we have embeddings for the *DGround* repositories using the corresponding feature set. So, we search and rank the repositories similar to each *QuerySet* repository. We extract five top matches for each query. Finally, we evaluate the performance of each feature set with the help of two metrics: (a) Similarity Precision, and (b) Success Rate.

Similarity Precision: Similarity Precision is defined as the proportion of highly similar and similar repositories in the ground-truth among the top 5 matches found by a feature set for a query.

TABLE II: Performance of different combinations of features.

Features	Similarity Precision(%)	Success Rate(%)
One Feature		
RepoName	51.0	71.3
Description	**66.0**	**85.1**
Topics	52.9	78.2
Readme	**90.1**	**98.5**
Two Features		
RepoName, Description	**74.7**	**91.6**
RepoName, Topics	67.8	90.1
RepoName, Readme	91.4	98.5
Description, Topics	74.4	92.1
Description, Readme	**93.7**	**99.5**
Topics, Readme	93.5	100
Three Features		
Description, Topics, Readme	**97.8**	**100**
RepoName, Topics, Readme	96.1	100
RepoName, Description, Readme	94.8	99.5
RepoName, Description, Topics	**80.8**	**95.0**

Success Rate: We define a query as successful if any of the top 5 matches by a feature set is found as similar/highly similar in the ground-truth. Success Rate is percentage of successful queries.

C. Results

We show the performance evaluation of the feature sets in Table II.

- *Readme is found as the most important feature. Readme* offers 90.1% Similarity Precision and 98.5% Success Rate, while *Readme* with *Description* increases Similarity Precision by 3.6% and Success Rate by 1.0%. While being small source of information compared to *Readme*, *RepoName/Topics* solely achieves 51%/52.9% Similarity Precision and 71.3%/78.2% Success Rate, *Description* provides 13% better Similarity Precision (66.0%) and 7% better Success Rate (85.1%).
- *RepoName and Description together as a feature set stands competitive considering its simplicity and availability*. *RepoName* and *Description* together achieves 74.7% Similarity Precision and 91.6% Success Rate. It is worth noting that adding *Readme* to any combination of feature set always increases both performance metrics. Accompanied by *Readme* with *RepoName* and *Description*, improves Precision and Success Rate by $\sim$ 20% and $\sim$ 8%, respectively.
- *Feature set of Description, Topics, and Readme shows the best performance among all combinations.*. This feature set provides 97.8% Similarity Precision and 100% Success Rate. Feature set of *Topics* and *Readme* offers highest Success Rate with any combination.

V. Discussion

We discuss the representativeness of our dataset in this section.

How representative is our data? We argue that our dataset is fairly representative since it contains repositories that

have significant exposure across the GitHub platform having at least 5 stars and 5 forks. These repositories gain such popularity because of the content and impact of the projects that they contribute to the community. Indeed, these influential repositories can be highly expected when mining repositories in different research domains.

Considering additional repository information. Having more information per repository is bound to provide additional depth in determining similarity but it will come at the computational cost of both: (a) collecting the data, and (b) comparing the repositories. One can consider four types of information: (a) metadata, including the author or the number of stars or forks of the repository, (b) user activity such as comments and issues, (c) social context, the people that follow the repository, the community of the author etc., and, of course, (d) the source code. Note that if the focus of the study is functional similarity, the specifics of the code implementation may be of less interest, when the metadata is available. We intend to explore the capabilities leveraging additional repository data in future work. However, downloading a full repository is a significantly resource consuming undertaking.

VI. Related Work

We can review prior related efforts into three categories.

a. Identifying similar GitHub repositories. Topic modeling techniques are used to categorize similar software repositories [15], [16]. CLAN [5] leverages API calls and RepoPal [17] utilizes metadata to infer similarity. CrossSim [18], [19] embeds the source code and different social interactions in a graph for repository similarity.

b. Repository representation in vector space. Repo2Vec [7] follows an embedding approach to determine repository similarity using metadata, source code and repo structure. On the other hand, Paper2Repo [20] recommends repositories given an academic paper based on their embedding. Later, Rep2Vec [8] constructs a repository heterogeneous graph to model the relationship among repository, user and topic which is then encoded using a graph neural network to generate repository embedding.

c. Mining software repositories. SourceFinder [21] collects GitHub repositories to identify repositories with malware source code. [11] creates a dataset of automotive software projects in GitHub.

VII. Conclusions

We propose MetaSim, an online search engine to find similar github repositories given a repository as query. Our experiment results show that though *Readme* is the most important feature offering 90.1% Similarity Precision and 98.5% Success Rate, feature set containing *RepoName*, *Description*, and *Topics* offers comparable 80.8% Similarity Precision and 95.0% Success Rate considering its simplicity as opposed to *Readme*. Our work can be seen as a building block for future work in the space of repository mining.

VIII. Acknowledgment

This work was supported by NSF SaTC grant No. 2132642.

References

[1] Y. Hu, J. Zhang, X. Bai, S. Yu, and Z. Yang, "Influence analysis of github repositories," *SpringerPlus*, vol. 5, no. 1, pp. 1–19, 2016.
[2] M. Gharehyazie, B. Ray, and V. Filkov, "Some from here, some from there: Cross-project code reuse in github," in *MSR*. IEEE, 2017, pp. 291–301.
[3] K. W. Nafi, B. Roy, C. K. Roy, and K. A. Schneider, "A universal cross language software similarity detector for open source software categorization," *Journal of Systems and Software*, vol. 162, p. 110491, 2020.
[4] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in GitHub for msr studies," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 560–564.
[5] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *ICSE*. IEEE, 2012, pp. 364–374.
[6] F. Thung, D. Lo, and L. Jiang, "Detecting similar applications with collaborative tagging," in *ICSM*. IEEE, 2012, pp. 600–603.
[7] M. O. F. Rokon, P. Yan, R. Islam, and M. Faloutsos, "Repo2Vec: A comprehensive embedding approach for determining repository similarity," in *ICSME*. IEEE, 2021, pp. 355–365.
[8] Y. Qian, Y. Zhang, Q. Wen, Y. Ye, and C. Zhang, "Rep2Vec: Repository embedding via heterogeneous graph adversarial contrastive learning," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.
[9] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the 11th Working Conference on MSR*, 2014.
[10] D. Gonzalez, T. Zimmermann, and N. 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.
[11] S. Kochanthara, Y. Dajsuren, L. Cleophas, and M. van den Brand, "Painting the landscape of automotive software in GitHub," in *MSR*, 2022.
[12] M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," *Empirical Software Engineering*, vol. 26, pp. 1–33, 2021.
[13] A. E. Rao and S. Chimalakonda, "Apples, oranges & fruits–understanding similarity of software repositories through the lens of dissimilar artifacts," in *ICSME*. IEEE, 2022, pp. 384–388.
[14] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.
[15] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Mudablue: An automatic categorization system for open source repositories," *Journal of Systems and Software*, vol. 79, no. 7, pp. 939–953, 2006.
[16] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent dirichlet allocation for automatic categorization of software," in *MSR*, 2009.
[17] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on GitHub," in *SANER*, 2017.
[18] P. T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio, "CrossSim: exploiting mutual relationships to detect similar oss projects," in *SEAA*, 2018.
[19] P. T. Nguyen, J. Di Rocco, R. Rubei, and Davide, "An automated approach to assess the similarity of github repositories," *Software Quality Journal*, vol. 28, pp. 595–631, 2020.
[20] H. Shao, D. Sun, J. Wu, Z. Zhang, A. Zhang, S. Yao, S. Liu, T. Wang, C. Zhang, and T. Abdelzaher, "paper2repo: Github repository recommendation for academic papers," in *The Web Conference 2020*, 2020.
[21] M. O. F. Rokon, R. Islam, A. Darki, E. E. Papalexakis, and M. Faloutsos, "SourceFinder: Finding malware source-code from publicly available repositories in github," in *RAID*, 2020, pp. 149–163.