A Dataset for GitHub Repository Deduplication

Diomidis Spinellis
Zoe Kotti
{dds,zoekotti}@aueb.gr
Athens University of Economics and Business

Audris Mockus
audris@mockus.org
University of Tennessee

ABSTRACT

GitHub projects can be easily replicated through the site's fork process or through a Git clone-push sequence. This is a problem for empirical software engineering, because it can lead to skewed results or mistrained machine learning models. We provide a dataset of 10.6 million GitHub projects that are copies of others, and link each record with the project's ultimate parent. The ultimate parents were derived from a ranking along six metrics. The related projects were calculated as the connected components of an 18.2 million node and 12 million edge denoised graph created by directing edges to ultimate parents. The graph was created by filtering out more than 30 hand-picked and 2.3 million pattern-matched clumping projects. Projects that introduced unwanted clumping were identified by repeatedly visualizing shortest path distances between unrelated important projects. Our dataset identified 30 thousand duplicate projects in an existing popular reference dataset of 1.8 million projects. An evaluation of our dataset against another created independently with different methods found a significant overlap, but also differences attributed to the operational definition of what projects are considered as related.

CCS CONCEPTS

• **Software and its engineering** → Open source model; **Software configuration management and version control systems**; • **General and reference** → *Empirical studies.*

KEYWORDS

Deduplication, fork, project clone, GitHub, dataset

ACM Reference Format:
Diomidis Spinellis, Zoe Kotti, and Audris Mockus. 2020. A Dataset for GitHub Repository Deduplication. In *17th International Conference on Mining Software Repositories (MSR '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3379597.3387496

1 INTRODUCTION

Anyone can create a copy of a GitHub project through a single effortless click on the project's *fork* button. Similarly, one can also create a repository copy with just two Git commands. Consequently, GitHub contains many millions of copied projects. This is a problem for empirical software engineering. First, when data containing multiple copies of a repository are analyzed, the results can end up skewed [27]. Second, when such data are used to train machine learning models, the corresponding models can behave incorrectly [2, 23].

```sql
select distinct p1, p2 from (
  select project_commits.project_id as p2,
    first_value(project_commits.project_id) over (
      partition by commit_id
      order by mean_metric desc) as p1
    from project_commits
    inner join forkproj.all_project_mean_metric
    on all_project_mean_metric.project_id =
         project_commits.project_id) as shared_commits
where p1 != p2;
```

Listing 1: Identification of projects with common commits

In theory, it should be easy to filter away copied projects. The project details provided by the GitHub API contain the field `fork`, which is *true* for forked projects. They also include fields under `parent` or `source`, which contain data concerning the fork source.

In practice, the challenges of detecting and grouping together copied GitHub repositories are formidable. At the computational level, they involve finding among hundreds of millions of projects those that are near in a space of billions of dimensions (potentially shared commits). The use of GitHub for courses and coursework with hundreds of thousands of participants,[1] for experimenting with version control systems,[2] and for all kinds of frivolous or mischievous activity[3] further complicates matters.

2 DATASET CREATION

An overview of the dataset's construction process is depicted in an extended version of this paper [39]. The projects were selected from GitHub by analyzing the GHTorrent [11, 13] dataset (release 2019-06-01) by means of the *simple-rolap* relational online analytical processing and *rdbunit* relational unit testing frameworks [14]. Following published recommendations [22], the code and primary data associated with this endeavor are openly available online,[4] and can be used to replicate the dataset or construct an updated version from newer data.

The GHTorrent dataset release we used contains details about 125 million (125 486 232) projects, one billion (1 368 235 072) individual commits, and six billion (6 251 898 944) commits associated with (possibly multiple, due to forks and merges) projects.

[1] https://github.com/rdpeng/ProgrammingAssignment2
[2] https://archive.softwareheritage.org/browse/search/?q=dvcsconnectortest&with_visit&with_content
[3] https://github.com/illacceptanything/illacceptanything
[4] https://doi.org/10.5281/zenodo.3742818

MSR '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7517-7/20/05…$15.00
https://doi.org/10.1145/3379597.3387496

We first grouped shared commits to a single "attractor" project, which was derived based on the geometric mean (Table *all project mean metric*—125 486 232 records) of six quality attributes:[5] recency of the latest commit (Table *most recent commit*—100 366 312 records), as well as the number of stars (Table *project stars*—10 317 662 records), forks (Table *project forks*—6 958 551 records), commits (Table *project ncommits*—100 366 312 records), issues (Table *project issues*—9 498 704 records), and pull requests (Table *project pull requests*—7 143 570 records). In addition, the project-id was used as a tie-breaker. To avoid the information loss caused by zero values [20], we employed the following formula proposed by del Cruz and Kref [9]:

$$G_{\epsilon,X}(X) = \exp\left(\frac{1}{n}\sum_{i=1}^{n}\log\left(x_i + \delta_*\right)\right) - \delta_*$$

with δ_* calculated to have the value of 0.001 for our data. We used a query utilizing the SQL window functions in order to group together shared commits (Table *projects sharing commits*—44 380 204 records) without creating excessively large result sets (see Listing 1).

To cover holes in the coverage of shared commits, we complemented the *projects sharing commits* table with projects related by GitHub project forks, after removing a set of hand-picked and heuristic-derived projects that mistakenly linked together unrelated clusters (Table *blacklisted projects*—2 341 896 records). In addition, we removed from the combined graph non-isolated nodes having between two and five edges, in order to reduce unwanted merges between unrelated shared commit and fork clusters. The method for creating the blacklisting table, along with the details behind the denoising process, are explained in Section 3.

We subsequently converted projects with shared commits or shared fork ancestry into an (unweighted) graph to find its connected components, which would be the groups of related projects. We identified connected components using the *GraphViz* [10] *ccomps* tool (Table *acgroups*—18 203 053 records). The final steps involved determining the size of each group (Table *group size*—2 472 758 records), associating it with each project (Table *project group size*—18 203 053 records) and its metrics (Table *project metrics*—18 203 053 records), obtaining the mean metric for the selected projects (Table *project mean metric*—18 203 053 records), and associating with each group the project excelling in the metric (Table *highest mean in group*—7 553 705 records). This was then used to create the deduplication table (Table *deduplicate by mean*—10 649 348 records) by partnering each project with a sibling having the highest mean value in the calculated metrics.

3 DOWN THE RABBIT HOLE

We arrived at the described process after numerous false starts, experiments, and considerable manual effort. Here we provide details regarding the technical difficulties associated with the dataset's creation, the rationale for the design of the adopted processing pipeline, and the process of the required hand-cleaning and denoising.

Our manual verification of large graph components uncovered a mega-component of projects with 4 278 791 members. Among the component's projects were many seemingly unrelated popular ones, such as the following ten: FreeCodeCamp/FreeCodeCamp, facebook/react, getify/You-Dont-Know-JS, robbyrussell/oh-my-zsh, twbs/bootstrap, Microsoft/vscode, github/gitignore, torvalds/linux, nodejs/node, and flutter/flutter.

We wrote a graph-processing script to remove from the graph all but one edges from projects with up to five edges. We chose five based on the average number of edges per node with more than one edge (3.8) increased by one for safety. We also looked at the effect of other values. Increasing the denoising limit up to ten edges reduced the size of the mega-component only little to 2 523 841. Consequently, we kept it at five to avoid removing too many duplicate projects. This improved somewhat the situation, reducing the size of the mega-component to 2 881 473 members.

Studying the mega-component we observed that many attractor projects were personal web sites.[G1,6] Focusing on them we found that apparently many clone a particular personal style builder, build on it, force push the commits, and then repeat the process with another builder project. For example, it seems that through this process, wicky-info/wicky-info.github.io shares commits with 139 other projects.

Based on this insight we excluded projects with names indicating web sites (all ending in github.io), and also removed from the graph nodes having between two and five edges, considering them as adding noise. This reduced considerably the mega-component size in the graph of projects with shared commits, to the point where the largest component consisted mostly of programming assignments forked and copied thousands of times (jtleek/datasharing — 199k forks, rdpeng/ProgrammingAssignment2 — 119k, rdpeng/RepData_PeerAssessment1 — 32.3k).

We also joined the generation of the fork tree and the common commit graph to reduce their interference, applying the denoising algorithm to both. This reduced the clusters to very reasonable sizes, breaking the mega-component to only include a few unrelated projects,[G2] which was further improved by blacklisting a couple of Android Open Source Project repositories.[G3]

We manually inspected the five projects with the highest mean ranking in each of the first 250 clusters, which comprise about 1.6 million projects. The most populous component (Linux) had 175 184 members and the last, least populous, component (vim) had 1912 members. Many cases of several high-ranked projects in the same component involved genuine forks. This is for example the case of MariaDB/server linking percona/percona-server, mysql/mysql-server, and facebook/mysql-8.0, among others.[G4] Where these referred to different projects, we drew a map of shortest path between the 49 top-ranked projects and the first or 50th one, and blacklisted low-ranked projects that were linking together unrelated repositories.

Resolved examples include the linking of Docker with Go,[G5] Django with Ruby on Rails,[G6] Google projects with zlib,[G7] Diaspora with Arduino,[G8] Elastic Search with Pandas,[G9] Definitely Types with RxJS,[G10] Ansible with Puppet,[G11] PantomJS with WebKit and Qt,[G12] OpenStack projects,[G13] Puppet modules,[G14] documentation projects,[G15] Docker registry with others,[G16] Drupal with Backdrop,[G17] Python and Clojure koans,[G18] Vimium with Hubot,[G19] as well as several ASP.NET projects.[G20]

[5] In the interest of readability, this text replaces the underscores in the table names with spaces.

[6] The referenced graph images GN are distributed with the paper's replication package.

For some clusters that failed to break up we repeated the exercise, looking at paths in the opposite direction, removing additional projects such as those linking Linux with Dagger,[G21] Ruby with JRuby, oh-my-zsh, Capistrano, and git-scm,[G22] and Laravel with Fuel.[G23]

In some cases the culprits were high-ranked projects, such as boostorg/spirit, which links together more than ten Boost repositories,[G7] apache/hadoop, which links with Intel-bigdata/SSM,[G24] DefinitelyTyped/DefinitelyTyped, which links to Reactive-Extensions/RxJS,[G10] ReactiveX/RxJava, which links several Netflix repositories,[G25] jashkenas/underscore, which links with lodash/lodash,[G26] jsbin/jsbin linking to cdnjs/cdnjs,[G27] ravendb/ravendb linking to SignalR/SignalR,[G28] Kibana linked with Grafana,[G29] CartoDB/carto linked with less/less.js.[G30] Other projects, such as Swift and LLVM,[G31] or Docker with Containerd,[G32] were too entangled to bring apart.

To further investigate what brings the component's projects together, we selected from the component one popular project with relatively few forks (creationix/nvm), and applied Dijkstra's shortest path algorithm to find how other projects got connected to it. We drew paths from that project to 30 other popular projects belonging to the same component, and started verifying each one by hand. We looked at the shared commits between unrelated projects that we found connected, such as yui-knk/rails and seuros/django.

Some (very few) commits appear to be shared by an inordinate number of projects. At the top, three commits are shared by 100 683 projects, another three by 67 280, and then four by 53 312. However, these numbers are not necessarily wrong, because there are five projects with a correspondingly large number of forks: 125 491 (jtleek/datasharing), 124 326 (rdpeng/ProgrammingAssignment2), 111 986 (octocat/Spoon-Knife), 70 137 (tensorflow/tensorflow), and 66 066 (twbs/bootstrap). The first two commits are associated with many (now defunct) projects of the user dvcsconnectortest (missingcommitsfixproof, missingcommitstest, and then missingcommitstest_250_1393252414399) for many different trailing numbers. However, the particular user is associated with very few commits, namely 1240, so it is unlikely that these commits have poisoned other components through transitive closure.

We later on improved the denoising to incorporate components that could be trivially determined for isolation, by looking at just the neighboring nodes. The algorithm we employed is applied to all nodes n having between two and five edges; the ones we used to consider as noise. It sums up as s being the number of edges of all nodes n' that were directly connected to n. If s is equal to the edges leading to n, then n and its immediate neighbors form a component, otherwise it is considered as adding noise and is disconnected from its neighbors. For a graph with edges E the condition for a node n being considered as noise, can be formally described as

$$\left|(n,n')|(n,n') \in E\right| \neq \sum_{\forall n' | (n,n') \in E} \left|\{(n',n'')|(n',n'') \in E\}\right|$$

Applying this algorithm decreased the number of ignored "noise projects" marginally from 37 660 040 to 37 333 119, increasing, as expected, the number of components by the same amount, from 2 145 837 to 2 472 758, and also increasing the number of projects considered as clones by about double that amount, from 9 879 677 to 10 649 348.

Table 1: Dataset Comparison

	Dataset	
Metric	CCFSC	CDSC
Number of repositories	10 649 348	116 265 607
Number of independent projects	2 470 126	63 829 733
Size of largest cluster	174 919	244 707
Average cluster size	4.3	1.8
Cluster size standard deviation	169	44
Reaper duplicates	30 095	80 079

4 DATASET OVERVIEW

The dataset is provided[7] as two files identifying GitHub repositories using the *login-name*/*project-name* convention. The file *deduplicate_names* contains 10 649 348 tab-separated records mapping a duplicated *source project* to a definitive *target project*. The file *forks_clones_noise_names* is a 50 324 363 member superset of the source projects, containing also projects that were excluded from the mapping as noise.

The files are to be used as follows. After selecting some projects for conducting an empirical software engineering study with GitHub projects, the first file should be used to map potentially duplicate projects into a set of definitive ones. Then, any remaining projects that appear in the second file should be removed as these are likely to be low-value projects with a high probability of undesirable duplication.

5 DUPLICATION IN EXISTING DATASETS

As an example of use of our dataset, we deduplicated the Reaper dataset [31], which contains scores concerning seven software engineering practices for about 1.8 million (1 853 205) GitHub projects. The study has influenced various subsequent works [1, 6, 8, 35] through the provided recommendations and filtering criteria for curating collected repositories. The authors have excluded deleted and forked projects, considering the latter as near duplicates.

Around 30 thousand (30 095) duplicate projects were identified in the Reaper dataset using *deduplicate_names*. The deduplication of the 800 hand-picked projects used in the classifiers' training and validation processes unveiled ten duplicate instances. Further investigation is required to measure any potential impact of the ten duplicate projects on the classification outcome. Nevertheless, researchers selecting projects from Reaper for their work can benefit from our dataset to filter out duplicate occurrences, to further improve the quality of selected projects and avoid the problems outlined in Section 1.

6 EVALUATION

We evaluated this dataset, which was constructed by identifying connected components based on forks and shared commits (CCFSC), through a quantitative and qualitative comparison with a similar dataset constructed using community detection of shared commits (CDSC) [30]. An overview of the basic characteristics of the two datasets appears in Table 1. The two datasets share a substantial overlap both in terms of source projects (8 157 317) and in terms of cluster leaders (5 513 580). On the other hand, it is clear that

[7] https://doi.org/10.5281/zenodo.3653920

CDSC is considerably more comprehensive than CCFSC in order of magnitude, covering more repositories. An important factor in its favor is that it covers other forges apart from GitHub, and therefore its population is a superset of CCFSC's. However, if one also considers the projects that CCFSC considers as noise (personal projects or projects with conflicting affiliations), the overlap swells to 40 338 421, covering about a third of the total. Furthermore, the fact that the increase in the Reaper dataset duplication in the CDSC dataset is only about double that of the CCFSC dataset indicates that the increased coverage of CCFSC may not be relevant for some empirical software engineering studies. These factors validate to some extent the dataset's composition.

To get a better understanding of where and how the two datasets vary, we also performed a qualitative evaluation. For this we selected a subgraph induced by the 1000 projects with the highest geometric mean score, and visualized the common and non-common elements of the 431 clusters that contained different nodes.[G33] In 301 cases the clusters shared at least one common element. The patterns we encountered mainly concern the following cases: CCFSC links more (and irrelevant) clusters compared to CDSC (e.g. FreeCodeCamp/FreeCodeCamp, gatsbyjs/gatsby, robbyrussell/oh-my-zsh); the converse happens (e.g. leveldb); CCFSC clusters related projects that CDSC does not cluster (e.g. tgstation/tgstation, bitcoin/bitcoin); the converse happens (e.g. hdl_qfs, t-s/blex); there is considerable agreement between the two (e.g. Homebrew/homebrew-core with afb/brew); there is considerable agreement but CDSC includes more related projects (e.g. aspnet/Mvc with h2h/Mvc). In general, we noticed that CDSC appears to be more precise at clustering than CCFSC, but worse at naming the clusters.

7 RELATED WORK

In distributed version control and source code management platforms, such as GitHub, developers usually collaborate using the pull request development model [12, 15–17], according to which repositories are divided into base and forked [25]. This constitutes one of the perils of mining GitHub: a repository is not necessarily a project [25], with commits potentially differing between the associated repositories.

Code duplication in GitHub was studied by Lopes et al. [27] through file-level and inter-project analysis of a 4.5 million corpus of non-forked projects. The overlap of files between projects, as given by the files' token hashes, was computed for certain thresholds and programming languages. JavaScript prevails with 48% of projects having at least 50% of files duplicated in other projects, and 15% of projects being 100% duplicated. Project-level duplication includes appropriations that could be addressed by Git submodules, abandoned derivative development, forks with additional non-source code content, and unorthodox uses of GitHub, such as unpushed changes. Code duplication can hamper the statistical reasoning in random selections of projects, and skew the conclusions of studies performed on them, because the observations (projects) are not independent, and diversity may be compromised. For the converse problem of obtaining similar GitHub repositories see the recent work by Phuong Nguyen and his colleagues [33] and the references therein.

While it is common sense to select a sample that is representative of a population, the importance of diversity is often overlooked, yet as important [4]. Especially in software engineering, where processes of empirical studies often depend on a large number of relevant context variables, general conclusions are difficult to extract [7]. According to Nagappan et al. [32], to provide a good sample coverage, selected projects should be diverse rather than similar to each other. Meanwhile, increasing the sample size does not necessarily increase generality when projects are not carefully selected.

Markovtsev and Kant in their work regarding topic modeling of public repositories using names in source code [29], recognized that duplicate projects contain few original changes and may introduce noise into the overall names distribution. To exclude them and accelerate the training time of the topic model, they applied Locality Sensitive Hashing [26] on the bag-of-words model. According to the analysis, duplicate repositories usually involve web sites, such as github.io, blogs and Linux-based firmwares, which align with our observations.

A duplication issue was also identified by Irolla and Dey [23] in the Drebin dataset [5], which is often used to assess the performance of malware detectors [18, 34] and classifiers [19, 38]. Half of the samples in the dataset have other duplicate repackaged versions of the same sequence of opcode. Consequently, a major part of the testing set may also be found in the training, inflating the performance of the designed algorithms. Experiments on classification algorithms trained on the Drebin dataset by including and excluding duplicates suggested moderate to strong underrated inaccuracy, and variation in the performance of the algorithms.

Similarly, Allamanis examined the adverse effects of code duplication in machine learning models of code [2]. By comparing models trained on duplicated and deduplicated code corpora, Allamanis concluded that performance metrics, from a user's perspective, may be up to 100% inflated when duplicates are included. The issue mainly applies to code completion [28, 37], type prediction [21, 36] and code summarization [3, 24], where models provide recommendations on new and unseen code.

8 RESEARCH AND IMPROVEMENT IDEAS

The main purpose of the presented dataset is to improve the quality of GitHub project samples that are used to conduct empirical software engineering studies. It would be interesting to see how such duplication affects published results by replicating existing studies after deduplicating the projects by means of this dataset. In addition, the dataset can be used for investigating the ecosystem of duplicated projects in terms of activity, duplication methods (forks vs commit pushes), tree depth, currency, or trustworthiness.

The dataset can be further improved by including projects from other forges and by applying more sophisticated cleaning algorithms.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825328.

REFERENCES

[1] Amritanshu Agrawal, Akond Rahman, Rahul Krishna, Alexander Sobran, and Tim Menzies. 2018. We Don't Need Another Hero? The Impact of "Heroes" on Software Development. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18)*. Association for Computing Machinery, New York, NY, USA, 245–253. https://doi.org/10.1145/3183519.3183549

[2] Miltiadis Allamanis. 2019. The Adverse Effects of Code Duplication in Machine Learning Models of Code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '19)*. Association for Computing Machinery, New York, NY, USA, 143–153. https://doi.org/10.1145/3359591.3359735

[3] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A Convolutional Attention Network for Extreme Summarization of Source Code. In *International Conference on Machine Learning (ICML '16)*. 2091–2100. https://arxiv.org/pdf/1602.03001.pdf

[4] Peter Allmark. 2004. Should Research Samples Reflect the Diversity of the Population? *Journal of medical ethics* 30 (May 2004), 185–189. https://doi.org/10.1136/jme.2003.004374

[5] Daniel Arp, Michael Spreitzenbarth, Hugo Gascon, and Konrad Rieck. 2014. Drebin: Effective and Explainable Detection of Android Malware in your Pocket. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS '14)*. The Internet Society. http://user.informatik.uni-goettingen.de/%7Ekrieck/docs/2014-ndss.pdf

[6] Sebastian Baltes and Stephan Diehl. 2019. Usage and Attribution of Stack Overflow Code Snippets in GitHub Projects. *Empirical Software Engineering* 24, 3 (June 2019), 1259–1295. https://doi.org/10.1007/s10664-018-9650-5

[7] Victor R. Basili, Forrest Shull, and Filippo Lanubile. 1999. Building Knowledge through Families of Experiments. *IEEE Trans. Softw. Eng.* 25, 4 (July 1999), 456–473. https://doi.org/10.1109/32.799939

[8] John Businge, Moses Openja, Sarah Nadi, Engineer Bainomugisha, and Thorsten Berger. 2018. Clone-Based Variability Management in the Android Ecosystem. In *Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME '18)*. 625–634. https://doi.org/10.1109/ICSME.2018.00072

[9] Roberto de la Cruz and Jan-Ulrich Kreft. 2018. Geometric mean extension for data sets with zeros. Available online https://arxiv.org/abs/1806.06403. arXiv:stat.AP/1806.06403

[10] Emden R. Gansner and Stephen C. North. 2000. An Open Graph Visualization System and its Applications to Software Engineering. *Software: Practice and Experience* 30, 11 (2000), 1203–1233. https://doi.org/10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.3.CO;2-E

[11] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 233–236. https://doi.org/10.5555/2487085.2487132

[12] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-Based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. Association for Computing Machinery, New York, NY, USA, 345–355. https://doi.org/10.1145/2568225.2568260

[13] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: Github's Data from a Firehose. In *9th IEEE Working Conference on Mining Software Repositories (MSR)*, Michele Lanza, Massimiliano Di Penta, and Tao Xie (Eds.). IEEE, 12–21. https://doi.org/10.1109/MSR.2012.6224294

[14] Georgios Gousios and Diomidis Spinellis. 2017. Mining Software Engineering Data from GitHub. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17)*. IEEE Press, Piscataway, NJ, USA, 501–502. https://doi.org/10.1109/ICSE-C.2017.164 Technical Briefing.

[15] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, 285–296. https://doi.org/10.1145/2884781.2884826

[16] Georgios Gousios and Andy Zaidman. 2014. A Dataset for Pull-Based Development Research. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*. Association for Computing Machinery, New York, NY, USA, 368–371. https://doi.org/10.1145/2597073.2597122

[17] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *Proceedings of the 37th International Conference on Software Engineering (ICSE '15)*, Vol. 1. 358–368. https://doi.org/10.1109/ICSE.2015.55

[18] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick D. McDaniel. 2017. On the (Statistical) Detection of Adversarial Examples. *ArXiv* abs/1702.06280 (2017).

[19] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. 2016. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *CoRR* abs/1606.04435 (2016). http://arxiv.org/abs/1606.04435

[20] Elsayed AE Habib. 2012. Geometric Mean for Negative and Zero Values. *International Journal of Research and Reviews in Applied Sciences* 11, 3 (2012), 419–32.

[21] Vincent J. Hellendoorn, Christian Bird, Earl T. Barr, and Miltiadis Allamanis. 2018. Deep Learning Type Inference. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18)*. Association for Computing Machinery, New York, NY, USA, 152–162. https://doi.org/10.1145/3236024.3236051

[22] Darrel C Ince, Leslie Hatton, and John Graham-Cumming. 2012. The Case for Open Computer Programs. *Nature* 482, 7386 (2012), 485–488.

[23] Paul Irolla and Alexandre Dey. 2018. The Duplication Issue within the Drebin Dataset. *Journal of Computer Virology and Hacking Techniques* 14, 3 (01 Aug. 2018), 245–249. https://doi.org/10.1007/s11416-018-0316-z

[24] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing Source Code using a Neural Attention Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 2073–2083. https://doi.org/10.18653/v1/P16-1195

[25] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2016. An In-Depth Study of the Promises and Perils of mining GitHub. *Empirical Software Engineering* 21, 5 (01 Oct. 2016), 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

[26] Jure Leskovec, Anand Rajaraman, and Jeff Ullman. 2020. *Chapter 3. Finding Similar Items.* Cambridge University Press, New York, NY, USA. http://infolab.stanford.edu/~ullman/mmds/book0n.pdf

[27] Cristina V. Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. 2017. DéjàVu: A Map of Code Duplicates on GitHub. *Proc. ACM Program. Lang.* 1, OOPSLA, Article Article 84 (Oct. 2017), 28 pages. https://doi.org/10.1145/3133908

[28] Chris J. Maddison and Daniel Tarlow. 2014. Structured Generative Models of Natural Source Code. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML '14)*. JMLR.org, 649–657.

[29] Vadim Markovtsev and Eiso Kant. 2017. Topic Modeling of Public Repositories at Scale using Names in Source Code. *CoRR* abs/1704.00135 (2017). arXiv:1704.00135 http://arxiv.org/abs/1704.00135

[30] Audris Mockus, Zoe Kotti, Diomidis Spinellis, and Gabriel Dusing. 2020. A Complete Set of Related Git Repositories Identified via Community Detection Approaches Based on Shared Commits. In *17th International Conference on Mining Software Repositories (MSR '20)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3379597.3387499

[31] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for Engineered Software Projects. *Empirical Software Engineering* 22, 6 (01 Dec 2017), 3219–3253. https://doi.org/10.1007/s10664-017-9512-6

[32] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in Software Engineering Research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '13)*. Association for Computing Machinery, New York, NY, USA, 466–476. https://doi.org/10.1145/2491411.2491415

[33] Phuong T. Nguyen, Juri Di Rocco, Riccardo Rubei, and Davide Di Ruscio. 2020. An automated approach to assess the similarity of GitHub repositories. *Software Quality Journal* (Feb. 2020). https://doi.org/10.1007/s11219-019-09483-0

[34] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 1–18. https://doi.org/10.1145/3132747.3132785

[35] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The Seven Sins: Security Smells in Infrastructure as Code Scripts. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, 164–175. https://doi.org/10.1109/ICSE.2019.00033

[36] Veselin Raychev, Martin Vechev, and Andreas Krause. 2015. Predicting Program Properties from "Big Code". In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. Association for Computing Machinery, New York, NY, USA, 111–124. https://doi.org/10.1145/2676726.2677009

[37] Veselin Raychev, Martin Vechev, and Eran Yahav. 2014. Code Completion with Statistical Language Models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*. Association for Computing Machinery, New York, NY, USA, 419–428. https://doi.org/10.1145/2594291.2594321

[38] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In *Research in Attacks, Intrusions, and Defenses*, Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 230–253.

[39] Diomidis Spinellis, Zoe Kotti, and Audris Mockus. 2020. A Dataset for GitHub Repository Deduplication: Extended Description. https://doi.org/10.5281/zenodo.3740595