A Study on Identifying Code Author from Real Development

Siyi Gong
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
gongsiyi@sjtu.edu.cn

Hao Zhong
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
zhonghao@sjtu.edu.cn

ABSTRACT

Identifying code authors is important in many research topics, and various approaches have been proposed. Although these approaches achieve promising results on their datasets, their true effectiveness is still in question. To the best of our knowledge, only one large-scale study was conducted to explore the impacts of related factors (*e.g.*, the temporal effect and the distribution of files per author). This study selected Google Code Jam programs as their subjects, but such programs are quite different from the source files that programmers write in daily development. To understand their effectiveness and challenges, we replicate their study and use their approach to analyze source files that are retrieved from real projects. The prior study claims that the temporal effect and the distribution of files per author have only minor impacts on their trained models. In the contrast, we find that in 85.48% pairs of training and testing sets, the accuracy of a trained model is less effective when the temporal effect is considered, and in total, the average accuracy decreases by 0.4298. In addition, when we use the real distribution of files as inputs, their approach can accurately identify only one or two core code authors, although a project can have more than ten authors. By revealing the limitations of the prior approach, our study sheds lights on where to make future improvements.

CCS CONCEPTS

• **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies** → *Natural language processing*; • **Software and its engineering** → *Software libraries and repositories.*

KEYWORDS

code authorship attribution, coding style evolution, empirical study

ACM Reference Format:
Siyi Gong and Hao Zhong. 2022. A Study on Identifying Code Author from Real Development. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3540250.3560878

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3560878

1 INTRODUCTION

Given a code snippet, the task of identifying its author is known as code authorship attribution. This task is important in various research topics (*e.g.*, bug report assignments [6], software forensics [29], and plagiarism detection [25]). For example, if the author of a piece of buggy code is identified, it is straightforward to assign the corresponding bug report to this code author. As another example, for software forensics, the identification of code authors is useful to trace malware samples. In the literature, researchers [5, 12, 33] have proposed various approaches to handle this task, but this task has many challenges. For example, the code style of an author can evolve over time [22]. Beside the temporal effect, the distribution of files can also affect their effectiveness [3]. To better understand these challenges, researchers [3] conducted a large-scale empirical study. According to their results, they conclude that the temporal effect on their accuracy is minor, and their approach [3] works well, when each code author has only five files in their dataset. Although their dataset includes programs from more than 8,000 authors, we notice that their settings are quite different from the real development. As a result, their findings can be compromised in the context of real development.

To meet the timely needs, in this paper, we conduct a replication study. In this study, we analyze a more recent version [4] of the approach that is used in the prior study [3]. Comparing with the prior study [3], our setting is closer to the real development (see Section 2 for details). For example, the prior study [3] mainly uses Google Code Jam programs that are submitted in two years, but we collect 14,657 commits from real projects over twenty years. As another example, we design more pairs of training and testing sets to analyze the impact of the time. This study explores the following research questions:

- **RQ1.** *What is the overall temporal effect on identifying code authors from real development?*
  After a model is trained, it inevitably becomes obsolete, since the styles of code authors can evolve over time. The prior study [3] shows that the impacts are minor. However, their analyzed source files are not from real development.
- **RQ2.** *What is the temporal effect on identifying individual code authors?*
  As most lines of code are written by several authors [19], some authors have insufficient files for mining. To explore the impacts, the prior study [3] reduces the files per author from nine to five. In their setting, authors have equal files, but when source files are retrieved from a real project, a few authors write much more files than others [19].

2 METHODOLOGY

This section introduces the methodology of our study.

Table 1: Our dataset.

Name	Time	Commit	Author	LOC
commons-collections	2001-2021	3,559	89	301,881
commons-dbcp	2004-2021	2,284	62	94,764
commons-configuration	2003-2021	3,347	38	269,997
commons-compress	2010-2021	2,493	65	87,177
commons-io	2002-2021	2,974	94	118,634
total	20	14,657	221	872,453

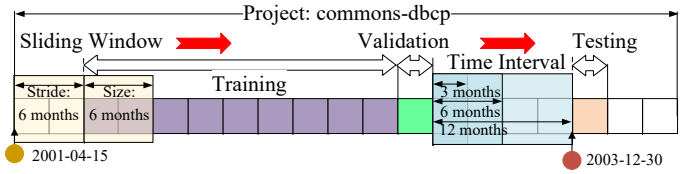


Figure 1: Our setting

Dataset. Table 1 shows our subject projects. Column "**Name**" lists project names. We choose these projects, since they have long maintenance histories. Column "**Time**" lists the maintained time of the project. In total, these projects have 14,657 commits that were submitted from 2001 to 2021. Column "**Commit**" lists the number of commits. As the prior studies [4, 19] did, from each commit, we extract its added lines as code snippets and assign the author of the commit as the label of these code snippets. Column "**Author**" lists the number of authors. Column "**LOC**" lists the lines of extracted code snippets.

From the Github code repository of each project, we extract all its commits. From each commit, we store its original and modified source files in a local directory. By comparing the original and modified versions of each file, we build a patch. A patch encodes a modification as an addition (+) and a deletion (-). In addition, for each commit, we record its commit information such as its author time, author name, author email, committer time, committer name, committer email and message. From each patch, we extract its added code lines as code snippets. The author name of the commit is considered as the author of the added lines, *e.g.*, the label of the code snippet, and the submission time is considered as the time of the code snippet.

The selected approach. In this study, we select a more recent version [4] of the approach that is used in the prior study [3]. This approach uses deep learning techniques (BiLSTM [28]) and is effective on their datasets (an accuracy of 86.41%). As Abuhamad *et al.* [4] did not release their tool, we implement the tool upon Keras [18]. In our study, we use our implementation to identify the programmers of Table 1. We select the identical parameters as Abuhamad *et al.* [4] did. Our implementation achieves an accuracy of 86.38%, which is close to what were reported in their paper.

Analysis overview. A prior study [3] has investigated the impact of the temporal effect and the files per author. Based on their results, they claim that both factors have only minor impacts on their approach. Table 2 lists their settings and ours. As shown in Column "**Dataset**", they used Google Code Jam programs [1]. Google Code Jam is a code competition. Although these programs are real code, they are different from the source files from a real project. First, most programs from GCJ are written by students, but not written by professional programmers. Second, the programs from GCJ resolve much simpler tasks than source files from a real project. Finally, programs from GCJ are written by individual authors, but source files from a real project are typically written by multiple code authors. To build a more realistic usage scenario, we use real projects to build our dataset. As explained in Column "**Author selection**", they required that authors must appear in both the training set and the testing set. When the authors in the testing set are unidentified, it is infeasible to determine whether they appear in the training set. As the requirement is unrealistic, we remove this requirement from our settings. As shown in Column "**Time strategy**", they used the programs from 2014 to 2015 as the training set and the programs from 2015 to 2016 as the testing set. The time interval between their training set and testing set is zero, and the impact of the time is not fully tested. Here, Table 3 of their paper [3] lists the programs of only two years, and Table 9 of their paper [3] lists only an accuracy value for each technique. They must present more values, if they tried more combinations. As shown in Column "**File strategy**", their dataset is still balanced and different from the real scenarios. As a comparison, we use the real projects where an author can have much more or fewer files.

3 EMPIRICAL RESULTS

More details are listed on our project website:
https://github.com/noonekowns/codeauthor

3.1 RQ1. The Overall Temporal Effect

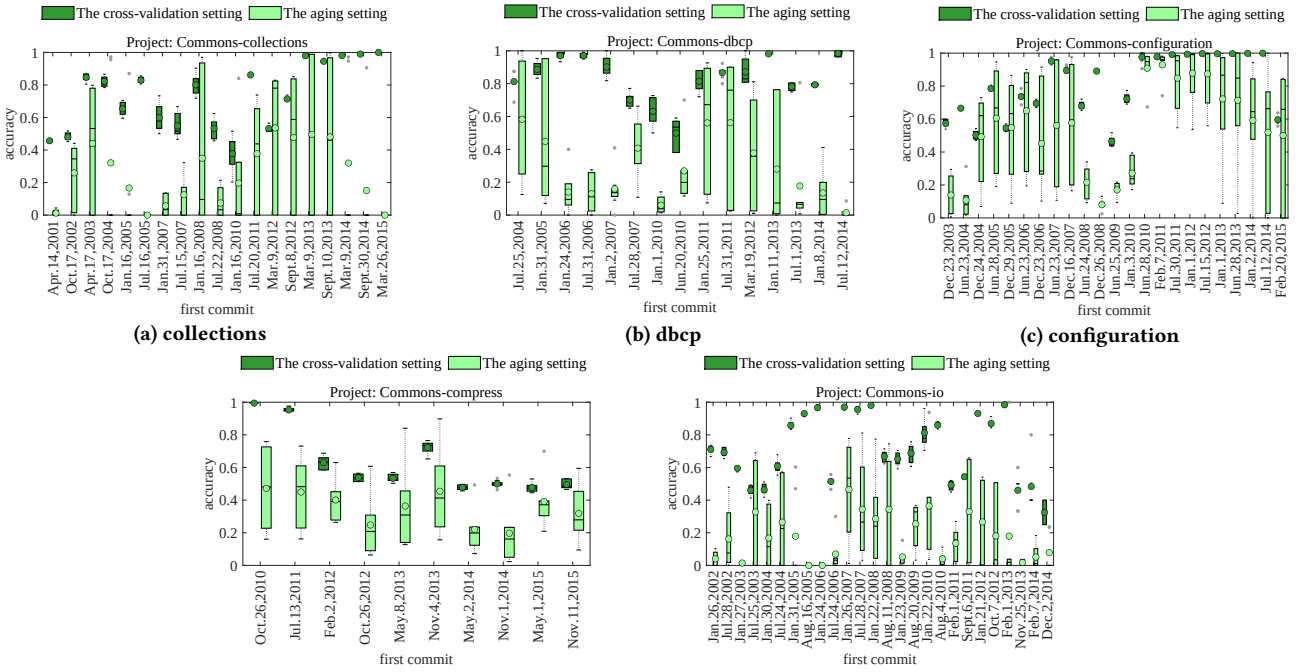
3.1.1 Protocol. As the baseline, we select the stratified 10-fold cross validation [26], since it is widely used in research papers [27, 31]. Comparing with the classical cross validation [10], the stratified cross validation uses the stratified sampling [26] to ensure that the training set and the testing set have the same proportion of labeled data as in the original dataset has. In each iteration, we split the submitted code snippets of a whole year into ten groups. In each fold, we use 9 groups as the training set and the remaining 1 group as the validation set. From each training set, we train 10 models, and we collect their validation accuracy values as the baseline, as the prior approaches [3, 33] did.

Figure 1 shows our settings to explore the impact of the time. To train each model, we use the commits of a whole year. For example, in Figure 1, the first commit and the last commit of the training set are submitted on Jan. 16, 2002 and Dec. 31, 2002, respectively. We use the same parameters with the baseline to train the model. A limitation of the prior study [3] is that the time interval between their training set and their testing set is zero, and their study does not fully explore the impacts when the interval increases. In our study, as shown in Figure 1, we increase the time interval from 3 months to 48 months, with three months as a gap. To obtain more pairs of training sets and testing sets, after each iteration, we apply a time sliding window. In particular, we move the whole training set to that of six months later. After all the models are trained and tested, we move the first commit of the training set to 6 months later, as shown in the sliding window of Figure 1. In total, we move the training set for several times until the training sets and testing sets cover the whole commits of each project.

3.1.2 Result. Figure 2 shows the box plots of the two settings. The horizontal axes list when the first commits of training sets are submitted. In each group, the cross-validation setting shows the

Table 2: The comparison between Abuhamad *et al.* [3] and our study.

Setting	Abuhamad *et al.* [3]	Our study
Dataset	Google Code Jam programs from 2014 to 2016	Real projects from 2001 to 2021
Author selection	Authors must appear in both the training set and the testing set	No requirement
Time strategy	A training set (from 2014 to 2015) and a testing set (from 2015 to 2016)	Much more combinations as shown in Figure 1
File strategy	The files per author are set to five, seven, and nine	The real distribution



(a) collections **(b) dbcp** **(c) configuration**

(d) compress **(e) io**

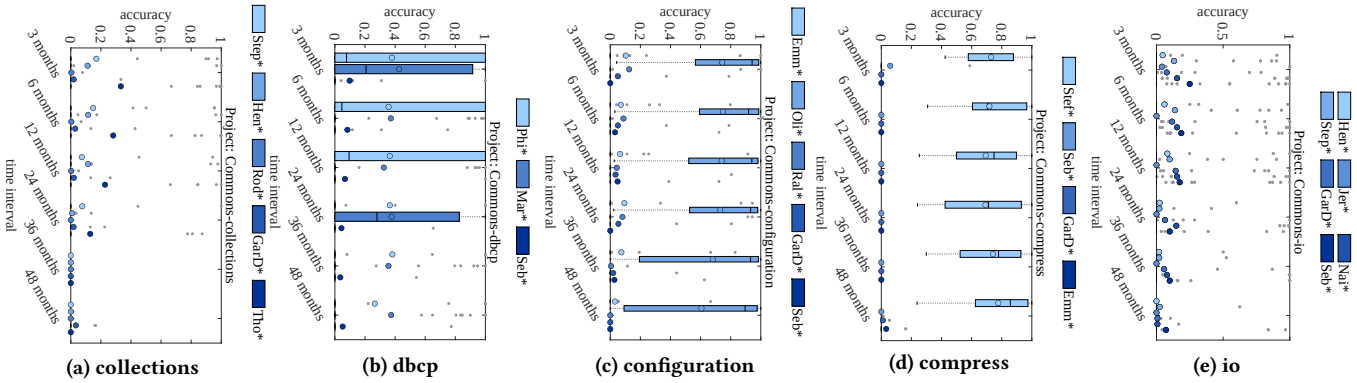
Figure 2: The overall temporal effect

accuracy values of the ten models that are trained from a training set, and the aging setting shows that the accuracy values when we increase the time interval. Except for the four exceptions in `configuration` and the one exception in `collections`, in 94.62% (88 out of 93) cases, the medians of the cross-validation setting are higher than those of the aging setting. We further calculate the average median reductions from the cross validation setting to the aging setting. The reductions are 0.5149 (`collections`), 0.5610 (`dbcp`), 0.2243 (`configuration`), 0.2989 (`compress`), and 0.5499 (`io`). For all the projects, the average reduction is 0.4298.

As shown in Figure 2, in several cases, the aging setting produces even better accuracy values than the cross-validation setting. For example, on Jun. 23, 2006 of `configuration`, the median of the cross-validation setting is even lower than that of the aging setting. We calculate the cases where the accuracy of the aging setting is more than the median of the cross-validation setting. The results are 15.79% (`collections`), 13.33% (`dbcp`), 25.36% (`configuration`), 11.67% (`compress`), and 5.77% (`io`). In total, such cases account for 14.52% of all cases. After inspecting those cases, we find that the distributions of authors lead to those extreme results. For example, for the data points on Jan. 31st, 2005 of `dbcp`, the training set mainly contains the code snippets of Dir* (58 lines of code) and Phi* (950 lines of code). After three months, the testing set mainly contains the code snippets of Dai* (2 lines of code) and Phi* (40 lines of code). The trained model blindly predicts most authors as Phi*, but still produces high accuracy, since the data are highly imbalanced. In more than 85.48% pairs of training and testing sets, the medians of accuracy become lower when the temporal effect is considered.

The prior study [3] takes the source files in a whole year as its training set. To eliminate the impact of the time period, we set the time period of our training data as one year. We increase the time interval between the training set and the testing set from 3 months to 12 months, with three months as a gap. Under this setting, we calculate the average median reductions from the cross validation setting to the aging setting. The reductions are 0.2442 (`collections`), 0.4164 (`dbcp`), 0.2000 (`configuration`), 0.3179 (`compress`), and 0.4029 (`io`). For all the projects, the average reduction is 0.3163. We also calculate the cases where the median accuracy of the aging setting is more than that of the cross-validation setting. The results are 28.07% (`collections`), 20.00% (`dbcp`), 33.33% (`configuration`), 6.67% (`compress`), and 11.54% (`io`). In total, such cases account for 21.15% of all cases. In more than 78.85% pairs of training and testing sets, the medians of accuracy become lower than those when the temporal effect is considered.

In summary, the average medians of accuracy values significantly decrease overtime, and the accuracy values decrease for most pairs of training and testing sets when the temporal effect is considered.



(a) collections (b) dbcp (c) configuration (d) compress (e) io

Figure 3: The temporal effect on individual authors

3.2 RQ2. The Effect on Individual Authors

3.2.1 Protocol. In this RQ, we explore the temporal effect on individual authors and analyze the impacts of files per author. To simulate the situation where each author has different files, the prior study [3] reduces the files per author from nine to five, but in the real development, code authors are not evenly distributed, and a few core programmers write most code lines [19]. In our study, we use the real distribution to train and test the models. After that, we record the accuracy for individual core members. Here, like the prior study [8], we consider a programmer as a core member of a project, if the time interval between his/her first and last commits in the project is larger than a year.

3.2.2 Result. Figure 3 shows the results of core authors. The horizontal axes list the time intervals. Although Figure 2 shows that the overall accuracy is high, for all the projects, we find that the trained models work well on only one or two authors. As the files per author are imbalanced, other measures (*e.g.*, MCC [13, 34]) are more suitable than the accuracy.

Except for `compress`, we find that the trained models become less effective when the time interval is larger. For example, in `dbcp`, when the time interval is 3-month, the trained model predicts the code of Phi* and Mar* accurately, but when it is 6-month, the trained model predicts accurately only the code of Phi*. We find that a trained model becomes less effective, when it is obsolete.

In summary, the trained models work well on only one or two authors, and their accuracy values largely decrease over time.

3.3 Threat to Validity

The threats to internal validity include wrong labels, since commits may not be submitted by their true authors. This threat is shared by the prior studies [4, 14], and its impact shall be minor, since Apache projects are carefully maintained. This threat could be further reduced by data sanitization techniques [9]. The threats to internal validity also include our implementation. The selected approach [4] is built upon TensorFlow [2], but we build the tool upon Keras. Despite of the differences, the impact shall be minor, since our effectiveness is similar with theirs. The threats to external validity include our limited subjects, although our subjects are already much more than the prior study [3]. This threat could be further reduced with more subjects.

4 WORK PLAN

In future work, we plan to conduct in-depth investigations on more challenges of identifying authors:

1. Exploring the impacts of the time on other approaches. In this study, we select only one subject, but researchers have proposed more approaches to identify code authors. Besides deep learning techniques [3, 11], other features (*e.g.*, n-grams [16], lexical features [7, 33], and Abstract Syntax Trees [5, 32]) and other techniques (*e.g.*, statistic analysis [12, 17]) are used to identify code authors. In future work, we plan to introduce more approaches and fully explore the impact of the time and the distribution.

2. Exploring the impacts of the cross-project predication. Besides what we explored in this study, there are other factors with practical values. For example, an author can write code for multiple projects. When an author joins a new project, this project may not contain sufficient histories for mining. Although it is feasible to use models trained from other projects, the team of a project can define quite different naming conventions and code styles from other teams [23]. As a result, a trained model may not work well on other projects. It is interesting to explore to what degree a trained model becomes less effective in this situation. We plan to explore these issues in our future work.

3. Learning features and techniques that are stable over time and across projects. From different features and techniques, we plan to explore those stable ones. Although the code style of an author inherently evolves over time and across projects, there can be some features that are more stable than others, and some techniques can be more robust to such changes. For example, although their approach identifies authors for natural language documents, Hansen *et al.* [21] analyzed how features evolve over time. Their ideas can motivate our work on identifying stable features and techniques. Besides identifying code authors, machine learning techniques [15, 30] and their variants [20, 24] have been used to resolve other software engineering problems. All these approaches suffer from the impacts of aging and cross-project learning, and our findings can be useful to improve these approaches.

ACKNOWLEDGMENTS

We appreciate reviewers for their insightful comments. Hao Zhong is the corresponding author. This work is sponsored by the CCF-Huawei Innovation Research Plan No. CCF2021-admin-270-202111.

REFERENCES

[1] 2019. Google Code Jam . https://codingcompetitions.withgoogle.com/codejam.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[3] Mohammed Abuhamad, Tamer AbuHmed, Aziz Mohaisen, and DaeHun Nyang. 2018. Large-scale and language-oblivious code authorship identification. In *Proc. CCS*. 101–114.

[4] Mohammed Abuhamad, Tamer Abuhmed, DaeHun Nyang, and David Mohaisen. 2020. Multi-χ: Identifying multiple authors from source code files. In *Proc. PETS*. 25–41.

[5] Bander Alsulami, Edwin Dauber, Richard Harang, Spiros Mancoridis, and Rachel Greenstadt. 2017. Source code authorship attribution using long short-term memory based networks. In *Proc. ESORICS*. 65–82.

[6] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proc. ICSE*. 361–370.

[7] Upul Bandara and Gamini Wijayarathna. 2013. Source code author identification with unsupervised feature learning. *Pattern Recognition Letters* 34, 3 (2013), 330–334.

[8] Lingfeng Bao, Xin Xia, David Lo, and Gail C Murphy. 2019. A large scale study of long-time contributor prediction for GitHub projects. *IEEE Transactions on Software Engineering* (2019), 1–22.

[9] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. 2009. The promises and perils of mining git. In *Proc. MSR*. 1–10.

[10] Michael W Browne. 2000. Cross-validation methods. *Journal of mathematical psychology* 44, 1 (2000), 108–132.

[11] Steven Burrows, Alexandra L Uitdenbogerd, and Andrew Turpin. 2014. Comparing techniques for authorship attribution of source code. *Software: Practice and Experience* 44, 1 (2014), 1–32.

[12] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing programmers via code stylometry. In *Proc. USENIX Security*. 255–270.

[13] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 1–13.

[14] Edwin Dauber, Aylin Caliskan, Richard Harang, and Rachel Greenstadt. 2018. Git blame who? stylistic authorship attribution of small, incomplete source code fragments. In *Proc. ICSE Companion*. 356–357.

[15] Felix Fischer, Huang Xiao, Ching-Yu Kao, Yannick Stachelscheid, Benjamin Johnson, Danial Razar, Paul Fawkesley, Nat Buckley, Konstantin Böttinger, Paul Muntean, et al. 2019. Stack overflow considered helpful! deep learning security nudges towards stronger cryptography. In *Proc. USENIX Security*. 339–356.

[16] Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis, Carole E Chaski, and Blake Stephen Howald. 2007. Identifying authorship by byte-level n-grams: The source code author profile (scap) method. *International Journal of Digital Evidence* 6, 1 (2007), 1–18.

[17] Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis, and Sokratis Katsikas. 2006. Effective identification of source code authors using byte-level information. In *Proc. ICSE*. 893–896.

[18] Aurélien Géron. 2019. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

[19] Siyi Gong and Hao Zhong. 2021. Code authors hidden in file revision histories: An empirical study. In *Proc. ICPC*. 71–82.

[20] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2019. Codekernel: A graph kernel based approach to the selection of api usage examples. In *Proc. ASE*. 590–601.

[21] Niels Dalum Hansen, Christina Lioma, Birger Larsen, and Stephen Alstrup. 2014. Temporal context for authorship attribution. In *Proc. IRFC*. 22–40.

[22] Vaibhavi Kalgutkar, Ratinder Kaur, Hugo Gonzalez, Natalia Stakhanova, and Alina Matyukhina. 2019. Code authorship attribution: Methods and challenges. *Comput. Surveys* 52, 1 (2019), 1–36.

[23] Ivan Krsul and Eugene H Spafford. 1997. Authorship analysis: Identifying the author of a program. *Computers & Security* 16, 3 (1997), 233–257.

[24] Yiling Lou, Qihao Zhu, Jinhao Dong, Xia Li, Zeyu Sun, Dan Hao, Lu Zhang, and Lingming Zhang. 2021. Boosting coverage-based fault localization via graph-based representation learning. In *Proc. ESEC/FSE*. 664–676.

[25] Alan Parker and James O Hamblen. 1989. Computer algorithms for plagiarism detection. *IEEE Transactions on Education* 32, 2 (1989), 94–99.

[26] Van L Parsons. 2014. Stratified sampling. *Wiley StatsRef: Statistics Reference Online* (2014), 1–11.

[27] S Madeh Piryonesi and Tamer E El-Diraby. 2020. Data analytics in asset management: Cost-effective prediction of the pavement condition index. *Journal of Infrastructure Systems* 26, 1 (2020), 04019036.

[28] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[29] Eugene H Spafford and Stephen A Weeber. 1993. Software forensics: Can we track code to its authors? *Computers & Security* 12, 6 (1993), 585–595.

[30] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *Proc. ASE*. 87–98.

[31] Brian H Willis and Richard D Riley. 2017. Measuring the statistical validity of summary meta-analysis and meta-regression results for use in clinical practice. *Statistics in medicine* 36, 21 (2017), 3283–3301.

[32] Wilco Wisse and Cor Veenman. 2015. Scripting dna: Identifying the javascript programmer. *Digital Investigation* 15 (2015), 61–71.

[33] Xinyu Yang, Guoai Xu, Qi Li, Yanhui Guo, and Miao Zhang. 2017. Authorship attribution of source code by using back propagation neural network based on particle swarm optimization. *PloS one* 12, 11 (2017), e0187204.

[34] Qiuming Zhu. 2020. On the performance of Matthews correlation coefficient (MCC) for imbalanced dataset. *Pattern Recognition Letters* 136 (2020), 71–80.