Enhancing the performance of software authorship attribution using deep autoencoders

Anamaria Briciu

Babeș-Bolyai University

WeADL 2023 Workshop

The workshop is organized under the umbrella of WeaMyL, project funded by the EEA and Norway Grants under the number RO-NO-2019-0133. Contract: No 26/2020.

June 9th 2023

Outline

1. Title
2. Introduction
   - The Authorship Attribution Task
   - Software AA (SAA)
3. SAA using autoencoders
   - The *AutoSoft* model
   - The *SoftId* model
4. Conclusions

Scientific Impact

- Gabriela Czibula, Mihaiela Lupea, Anamaria Briciu – *"Enhancing the performance of software authorship attribution using an ensemble of deep autoencoders"*, Mathematics, Special Issue "Recent Advances in Artificial Intelligence and Machine Learning", 2022, 10(15):2572
- Mihaiela Lupea, Anamaria Briciu, Istvan-Gergely Czibula, Gabriela Czibula – *"SoftId: An autoencoder-based one-class classification model for software authorship identification"*, 26th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2022), September 7-9, 2022, Procedia Computer Science 207, pp. 716-725

Authorship attribution

Definition: Authorship attribution (AA) is the task of determining the likely author of a given text

Importance of domain: wide range of applications in:

- literature and history
- education
- social network analysis
- software engineering and cybersecurity

Software authorship attribution

Identify the author of a code fragment.

Problem relevance

Software authorship identification applications in software development: *software quality*, *legacy software systems*, *software archaeology*, *fraud/plagiarism detection activities in education*

In **software engineering**:

- practical use in multiple scenarios
- e.g. maximize the benefit of the code review process given time and other constraints by using AA model to select or prioritize code to review

Challenges

- code reuse, development of a program by a *team* of developers, structural and layout characteristics altered by code formatters

Motivation

- develop flexible, adaptable models of authorship attribution
- **Idea:** develop AA & SAA models based on *autoencoders*
- **Goal:**
  - draw on natural language techniques and models in order to propose novel methods for authorship attribution of software
  - build efficient and general models that can be used in a variety of contexts

Software authorship attribution: Existing work

- **Features**: typographic or layout characteristics of programs [OC89], source code *N*-grams [BT07, FSGK06, Ten13], syntactic features (based on Abstract Syntax Trees (ASTs)) [UJAT20, ADH+17], learned "deep" representations [ARA+19]
- **Algorithms**: SVM [RZM11], LSTM and BiLSTM [AAMN18, ADH+17], CNN [ARA+19]
- **using LSI/TF-IDF**: [MM00] (*identify similarities between pieces of source code*), [BVE15] (*detect semantic re-implementations*)
- **using autoencoders**: [STASH19] *(task: authorship verification; domain: cybercrime; texts: IRC messages; deep AE as one-class classifier)*, [MY07] *(AE-based one-class classification model for document retrieval task)*

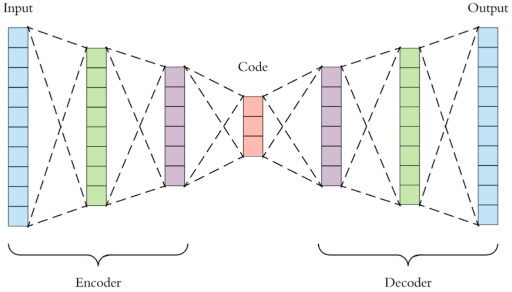
Original contributions

Two *autoencoder*-based models:

- *AutoSoft*: **Multi-class** software authorship attribution
- *SoftId*: **One-class** software authorship attribution

Autoencoders (AE)

- deep learning models used in medical data analysis, image analysis, bioinformatics and other fields
- self-supervised learning technique
- the goal is to extract meaningful features while encoding, having a representative code from which the input can be reconstructed



AutoSoft: Software authorship attribution

1. Formalization of the SAA problem
2. The *AutoSoft* model
3. Data set description
4. *AutoSoft* results
5. The $AutoSoft^{ext}$ model
6. $AutoSoft^{ext}$ results
7. Discussion

Formalization of the SAA problem

A multi-class classification problem.

- set of developers $\mathcal{DEV} = \{Dev_1, Dev_2, \ldots Dev_n\}$
- set of software programs $\mathcal{SP} = \{sp_1, sp_2, \ldots sp_r\}$
- **GOAL:** approximate a target function $f : \mathcal{SP} \rightarrow \mathcal{DEV}$ that maps a software program sp from $\mathcal{SP}$ to a certain class/developer $dev \in \mathcal{DEV}$

The *AutoSoft* model

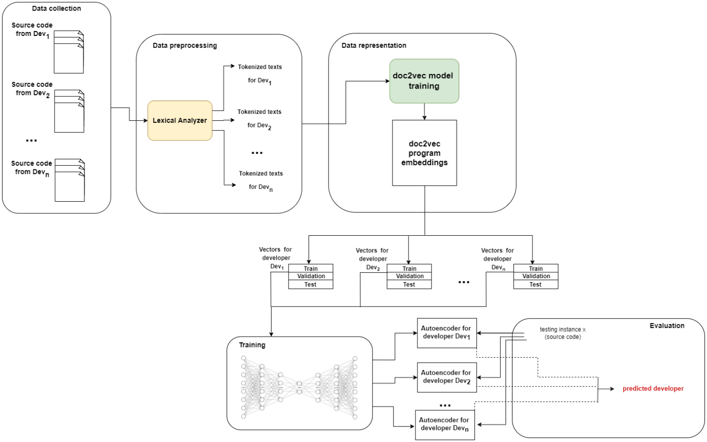


Figure: Overview of *AutoSoft*.

Dataset description

Considered data: subset of the Google Code Jam[2] data set.
Programming language: Python.

	Subset		
	5 developers	**12 developers**	**87 developers**
No. of files per developer	≥ 200	≥ 150	≥ 100
Total. no. files	1 132	2 357	11 089
Total. no. tokens	799 824	1 395 560	4 563 661
Median tokens per file	378.5	386	309
Median lines per file	61	65	52
Avg. no. tokens per file	706.56	592.09	411.55
Avg. no. lines per file	60.95	75.51	61.43

Table: Data set description (GCJ subsets)

[2] https://codingcompetitions.withgoogle.com/codejam

Results (I)

	Number of features	Performance measure	*N*-gram size				
			1	3	5	6	8
5 developers	150	*Precision*	0.984±0.008	**0.993**±0.004	0.989±0.007	0.988±0.006	0.988±0.005
		Recall	0.982 ±0.009	**0.993** ±0.004	0.988 ±0.009	0.987±0.008	0.988±0.005
		*F*1	0.983±0.008	**0.993**±0.004	0.988±0.009	0.987±0.008	0.988±0.005
	300	*Precision*	0.986±0.007	0.984±0.006	0.985±0.007	0.991±0.007	**0.992**±0.005
		Recall	0.986±0.007	0.98±0.008	0.985±0.007	0.991±0.007	**0.992**±0.005
		*F*1	0.986±0.007	0.98±0.005	0.985±0.007	0.991±0.007	**0.992**±0.005
12 developers	150	*Precision*	0.968±0.006	0.98±0.005	**0.984**±0.005	0.98±0.007	0.973±0.006
		Recall	0.966 ±0.007	0.979 ±0.005	**0.982** ±0.006	0.978±0.007	0.97±0.007
		*F*1	0.966±0.007	0.979±0.005	**0.982**±0.006	0.978±0.007	0.97±0.007
	300	*Precision*	0.977±0.007	**0.984**±0.007	0.98±0.004	0.979±0.005	0.978±0.008
		Recall	0.975±0.007	**0.981**±0.008	0.978±0.005	0.977±0.006	0.977±0.008
		*F*1	0.975±0.007	**0.981**±0.008	0.979±0.005	0.977±0.006	0.977±0.008
87 developers	150	*Precision*	0.882±0.004	0.892±0.004	**0.913**±0.004	0.911±0.004	0.906±0.006
		Recall	0.868 ±0.005	0.88±0.004	**0.901** ±0.005	0.899±0.004	0.895±0.007
		*F*1	0.866±0.005	0.88±0.004	**0.898**±0.005	0.896±0.004	0.889±0.008
	300	*Precision*	0.913±0.003	0.918±0.006	**0.922**±0.004	0.914±0.004	0.904±0.007
		Recall	0.902±0.003	0.911±0.005	**0.913**±0.004	0.905±0.006	0.894±0.007
		*F*1	0.902±0.003	0.909±0.007	**0.913**±0.004	0.904±0.005	0.89±0.005

Table: *AutoSoft* results with respect to *N*-gram size for subsets of 5, 12 and 87 developers. 95% confidence intervals are used for the results.

Results (II)

Type of features	Number of authors	Classifiers				
		AutoSoft	SVC	RF	GNB	kNN
unigrams	5	0.986	**0.993**	0.981	0.963	0.975
	12	0.975	**0.993**	0.965	0.946	0.958
	87	0.902	**0.953**	0.735	0.841	0.854
trigrams	5	0.98	**0.994**	0.98	0.972	0.96
	12	0.981	**0.992**	0.976	0.947	0.936
	87	0.909	**0.953**	0.734	0.855	0.817
5-grams	5	0.985	**0.998**	0.982	0.971	0.99
	12	0.979	**0.99**	0.976	0.935	0.983
	87	0.913	**0.95**	0.785	0.835	0.909
WIN				**25**		
LOSE				11		

Table: Comparison between *AutoSoft* and classifiers from the literature in terms of *F-score*.

Discussion

- *AutoSoft* obtains good performance in the task of authorship attribution, comparing favorably to existing classifiers
- the `doc2vec` representation manages to capture author particularities
- N-gram features with $N > 1$ perform better than simple unigrams, but no universally benefic value for N can be identified

The *AutoSoft*^{ext} model

- extension of *AutoSoft* to recognize not only the set of original developers on which it was trained, but an “unknown” class as well
- **classification stage**: prior step to multi-class classification: decide the likelihood that a software program sp belongs to the “unknown” class.
- likelihood computed using loss-based *distances* between a test instance and the given autoencoders

$$p_{unknown}(sp) = 0.5 + \frac{\prod_{i=1}^{j} dist_i(sp)}{2 \cdot \prod_{i=1}^{j} (l_i(sp) + \tau_i)}, \tag{1}$$

Results

Developer	*N*-gram type	Performance measures				
		Accuracy	*Precision*	*Recall*	*F*1	*Specificity*
Dev_{u1}	unigrams	0.904 ±0.015	0.926±0.012	0.965±0.013	0.945± 0.009	0.537±0.082
	5-grams	**0.974±0.006**	**0.977±0.007**	**0.994±0.005**	**0.993±0.003**	**0.858±0.046**
Dev_{u2}	unigrams	0.921 ±0.014	0.947±0.01	0.965±0.013	0.956± 0.008	0.587±0.087
	5-grams	**0.988±0.006**	**0.993±0.004**	**0.994±0.005**	**0.993±0.003**	**0.947±0.033**
Dev_{u3}	unigrams	0.891 ±0.011	0.917±0.006	0.965±0.013	0.94± 0.007	0.333±0.055
	5-grams	**0.979±0.009**	**0.983±0.01**	**0.994±0.005**	**0.988±0.005**	**0.867±0.083**
Dev_{u4}	unigrams	0.89 ±0.016	0.911±0.01	0.965±0.013	0.937± 0.009	0.4±0.074
	5-grams	**0.977±0.008**	**0.98±0.007**	**0.994±0.005**	**0.987±0.005**	**0.872±0.046**
Dev_{u5}	unigrams	0.871 ±0.011	0.896±0.007	0.965±0.013	0.929± 0.007	0.2±0.063
	5-grams	**0.965±0.009**	**0.968±0.01**	**0.994±0.005**	**0.981±0.005**	**0.762±0.079**
Dev_{u6}	unigrams	0.924 ±0.017	0.948±0.009	0.965±0.013	0.956± 0.01	0.679±0.054
	5-grams	**0.992±0.005**	**0.997±0.003**	**0.994±0.005**	**0.996±0.003**	**0.984±0.016**
Dev_{u7}	unigrams	0.899 ±0.014	0.922±0.011	0.965±0.013	0.943± 0.008	0.483±0.078
	5-grams	**0.97±0.009**	**0.972±0.007**	**0.994±0.005**	**0.983±0.003**	**0.817±0.046**

Table: *AutoSoft*^{ext} results with respect to *N*-gram size for 7 "unknown" authors and an $\mathcal{O}$*riginal* set with $n = 5$. 95% confidence intervals are used for the results.

Discussion

- *AutoSoft*^{ext} can be used to recognize whether given test instances belong to a considered group of developers, or to some other "unknown" developer
- when tested against existing one-class classification models such as One-Class Support Vector Machines, *AutoSoft*^{ext} compares favorably

SoftId: Software authorship attribution

1. Formalization of the SAA one-class classification problem
2. Overview of *SoftId*
3. Data set description
4. Results
5. Discussion

Formalization of the SAA problem

SAA as one-class classification problem

- set of k known software developers (authors) $\mathcal{S}d = \{Sd_1, Sd_2, \dots Sd_k\}$
- set of software programs $\mathcal{SC} = Sc_1 \cup Sc_2 \cup \cdots \cup Sc_k$
- **GOAL:** approximate a target function $t : \mathcal{SC} \to \{``+", ``-"\}$ that maps a software code $sc \in \mathcal{SC}$ to the *positive* class (formed by the developers from $\mathcal{S}d$) or the *negative* ("other") one

Overview of *SoftId*

Algorithm Classification for the testing source code sc.

function $\textsc{Classify}(\mathcal{S}d, A, sc)$

Require:

$\mathcal{S}d$ - the set of original software developers; A - the AE trained to recognize the developers from $\mathcal{S}d$;

sc - the testing instance (source code) to be classified

Ensure:

return the predicted class ("original" or "other")

$vec_{sc} \leftarrow$ the vector representation of sc

$p_{other}(sc) = 0.5 + \frac{D(vec_{sc}, \widehat{vec_{sc}}) - \tau}{2 \cdot (D(vec_{sc}, \widehat{vec_{sc}}) + \tau)}$ /* Compute the probability that sc belongs to the "other" class*/

if $p_{other}(sc) \geq 0.5$ **then**

$c \leftarrow$ "other"

else

$c \leftarrow$ "original"

end if

return c

end function

Data set description

- subsets of the **GCJ** data set
  - 3 “original” developers (709 software programs)
  - 5 “original” developers (1110 software programs)
  - 12 “original” developers (2325 software programs)
- randomly sampled “other” instances from the rest of the data set

Results (I)

No. of original authors	N-grams	TF-IDF representation							LSI representation						
		Acc	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*	*Acc*	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*
3	**5-grams**	0.947 ±0.006	1.000 ± 0.000	0.941 ± 0.007	0.970 ± 0.003	1.000 ± 0.000	**0.971** ± 0.003	0.971 ± 0.003	0.932 ±0.012	1.000 ±0.000	0.926 ±0.013	0.961 ±0.007	1.000 ±0.000	0.963 ±0.007	0.963 ±0.07
	6-grams	0.943 ±0.010	1.000 ± 0.000	0.937 ± 0.011	0.967 ± 0.006	1.000 ± 0.000	0.969 ± 0.006	0.969 ± 0.006	0.936 ±0.013	1.000 ±0.000	0.930 ±0.015	0.964 ±0.008	1.000 ±0.000	**0.965** ±0.007	0.965 ±0.007
	8-grams	0.939 ±0.014	1.000 ± 0.000	0.933 ± 0.016	0.965 ± 0.009	1.000 ± 0.000	0.966 ± 0.008	0.966 ± 0.008	0.936 ±0.016	1.000 ±0.000	0.930 ±0.017	0.964 ±0.010	1.000 ±0.000	**0.965** ±0.009	0.965 ±0.009
	10-grams	0.926 ±0.015	1.000 ± 0.000	0.919 ± 0.017	0.957 ± 0.009	1.000 ± 0.000	0.959 ± 0.008	0.959 ± 0.008	0.923 ±0.013	1.000 ±0.000	0.916 ±0.015	0.956 ±0.008	1.000 ±0.000	0.958 ±0.007	0.958 ±0.007
5	**5-grams**	0.943 ±0.013	0.995 ±0.002	0.943 ±0.014	0.968 ±0.007	0.950 ±0.016	0.946 ±0.015	0.969 ±0.008	0.929 ±0.015	0.999 ±0.001	0.923 ±0.016	0.959 ±0.009	0.988 ±0.007	0.955 ±0.012	0.961 ±0.008
	6-grams	0.941 ±0.017	0.994 ±0.001	0.940 ±0.018	0.966 ±0.010	0.947 ±0.010	0.944 ±0.014	0.967 ±0.010	0.933 ±0.020	0.998 ±0.001	0.928 ±0.022	0.962 ±0.012	0.982 ±0.008	0.955 ±0.015	0.963 ±0.011
	8-grams	0.947 ±0.013	0.996 ±0.001	0.945 ±0.014	0.970 ±0.007	0.962 ±0.009	**0.954** ±0.012	0.971 ±0.007	0.934 ±0.016	0.999 ±0.001	0.928 ±0.018	0.962 ±0.010	0.988 ±0.005	**0.958** ±0.012	0.963 ±0.009
	10-grams	0.937 ±0.013	0.996 ±0.001	0.935 ±0.015	0.964 ±0.008	0.964 ±0.010	0.949 ±0.012	0.965 ±0.008	0.921 ±0.014	0.998 ±0.001	0.915 ±0.015	0.954 ±0.008	0.983 ±0.005	0.949 ±0.010	0.956 ±0.008
12	**5-grams**	0.915 ±0.007	0.965 ±0.003	0.941 0.008	0.953 ±0.004	0.656 ±0.027	0.798 ±0.017	0.953 ±0.005	0.902 ±0.008	0.979 ±0.002	0.912 ±0.009	0.944 ±0.005	0.798 ±0.016	0.855 ±0.013	0.945 ±0.005
	6-grams	0.914 ±0.008	0.971 ±0.002	0.934 ±0.009	0.952 ±0.005	0.716 ±0.021	0.825 ±0.015	0.952 ±0.005	0.903 ±0.010	0.982 ±0.002	0.910 ±0.011	0.945 ±0.006	0.834 ±0.017	**0.872** ±0.014	0.946 ±0.006
	8-grams	0.904 ±0.012	0.976 ±0.002	0.917 ±0.013	0.945 ±0.007	0.772 ±0.020	0.845 ±0.017	0.946 ±0.008	0.912 ±0.009	0.981 ± 0.002	0.920 ±0.009	0.950 ±0.006	0.823 ±0.023	0.871 ±0.016	0.951 ±0.006
	10-grams	0.873 ±0.011	0.982 ±0.002	0.877 ±0.012	0.926 ±0.007	0.838 ±0.016	**0.857** ±0.014	0.929 ±0.007	0.874 ±0.011	0.980 ±0.003	0.880 ±0.011	0.927 ±0.007	0.820 ±0.023	0.850 ±0.017	0.930 0.007

Table: Performance metrics obtained by evaluating *SoftId* classifier on the Google Code Jam data set. 95% CI are used for the results.

Results (II)

No. of original authors	N-grams	TF-IDF representation							LSI representation						
		Acc	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*	*Acc*	*Prec*	*Recall*	*F1*	*Spec*	*AUC*	*AUPRC*
3	**5-grams**	0.043	0.008	0.040	0.025	0.071	0.056	0.024	0.038	0.011	0..031	0.023	0.107	0.069	0.021
	6-grams	0.058	0.008	0.056	0.035	0.077	0.066	0.032	0.058	0.011	0.053	0.035	0.104	0.079	0.032
	8-grams	0.039	0.000	0.043	0.024	0.000	0.021	0.021	0.044	0.003	0.046	0.027	0.027	0.036	0.024
	10-grams	0.043	0.000	0.047	0.027	0.000	0.024	0.024	0.044	0.010	0.039	0.027	0.099	0.069	0.025
5	**5-grams**	0.072	0.022	0.060	0.042	0.196	0.128	0.041	0.077	0.042	0.045	0.044	0.388	0.217	0.044
	6-grams	0.075	0.015	0.070	0.045	0.129	0.100	0.042	0.085	0.021	0.075	0.051	0.185	0.130	0.048
	8-grams	0.075	0.000	0.083	0.046	-0.005	0.039	0.041	0.065	0.011	0.062	0.040	0.095	0.078	0.036
	10-grams	0.077	-0.001	0.086	0.048	-0.012	0.037	0.043	0.069	0.016	0.062	0.042	0.141	0.101	0.039
12	**5-grams**	0.072	0.031	0.050	0.041	**0.292**	**0.171**	**0.041**	0.085	0.063	0.032	0.047	**0.619**	**0.325**	**0.047**
	6-grams	0.077	0.031	0.057	0.045	0.282	0.169	**0.044**	0.071	0.043	0.039	0.041	0.398	0.218	0.041
	8-grams	0.058	0.018	0.048	0.034	0.155	0.102	0.033	0.078	0.027	0.061	0.046	0.241	0.151	0.044
	10-grams	0.059	0.011	0.056	0.037	0.087	0.071	0.034	0.071	0.020	0.061	0.044	0.173	0.117	0.041

Table: Improvement achieved by *SoftId* classifier compared to *OSVM*.

Discussion

- good performance of *SoftId* with both representations
- decreasing *AUC* value as the number of developers increases (mainly due to *Specificity*)
- ideal *N*-gram value dependent on the testing context
- LSI representation generates better results than TF-IDF for larger corpora
- *SoftId* brings clear improvement over *OSVM* (in 95% cases)
- a tool like *SoftId* can be important in the software development process of projects inside small teams (3-12 developers).

Conclusions

- the two autoencoder-based models proposed obtained good performances on tasks of SAA
- the developed models are general, and highly adaptable
- **future work**: carry out experiments on data sets collected from software development teams

Q&A time!

Bibliography

Mohammed Abuhamad, Tamer AbuHmed, Aziz Mohaisen, and DaeHun Nyang.
Large-scale and language-oblivious code authorship identification.
In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 101–114, 2018.

Bander Alsulami, Edwin Dauber, Richard Harang, Spiros Mancoridis, and Rachel Greenstadt.
Source code authorship attribution using long short-term memory based networks.
In *European Symposium on Research in Computer Security*, pages 65–82. Springer, 2017.

Mohammed Abuhamad, Ji-su Rhim, Tamer AbuHmed, Sana Ullah, Sanggil Kang, and DaeHun Nyang.
Code authorship identification using convolutional neural networks.
Future Generation Computer Systems, 95:104–115, 2019.

Steven Burrows and Seyed M. M. Tahaghoghi.
Source code authorship attribution using n-grams.
In *Proceedings of the twelth Australasian document computing symposium, Melbourne, Australia, RMIT University*, pages 32–39, 2007.

Veronika Bauer, Tobias Volke, and Sebastian Eder.
Comparing TF-IDF and LSI as IR technique in an approach for detecting semantic re-implementations in source code.
Project code Software Campus, TU Munchen, grant number 01IS12057, 2015.

Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis, and Sokratis Katsikas.
Source code author identification based on n-gram author profiles.
In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 508–515. Springer, 2006.

Jonathan I. Maletic and Andrian Marcus.
Using Latent Semantic Analysis to identify similarities in source code to support program understanding.
In *Proceedings 12th IEEE internationals conference on tools with artificial intelligence. ICTAI 2000*, pages 46–53. IEEE, 2000.

Larry Manevitz and Malik Yousef.
One-class document classification via neural networks.
Neurocomputing, 70(7-9):1466–1481, 2007.

Paul W. Oman and Curtis R. Cook.
Programming style authorship analysis.
In *Proceedings of the 17th conference on ACM Annual Computer Science Conference*, pages 320–326, 1989.

Nathan Rosenblum, Xiaojin Zhu, and Barton P. Miller.
Who wrote this code? Identifying the authors of program binaries.
In *European Symposium on Research in Computer Security*, pages 172–189. Springer, 2011.

Sicong Shao, Cihan Tunc, Amany Al-Shawi, and Salim Hariri.
One-class Classification with Deep Autoencoder Neural Networks for Author Verification in Internet Relay Chat.
In *Proceedings of 16th IEEE/ACS International Conference on Computer Systems and Applications*, pages 1–8, 2019.

Matthew F. Tennyson.

A replicated comparative study of source code authorship attribution.
In *2013 3rd International Workshop on Replication in Empirical Software Engineering Research*, pages 76–83, 2013.

Farhan Ullah, Sohail Jabbar, and Fadi AlTurjman.
Programmers‘ de-anonymization using a hybrid approach of abstract syntax tree and deep learning.
Technological Forecasting and Social Change, 159:120186, 2020.