The Promise and Perils of Using Machine Learning When Engineering Software (Keynote Paper)

Yuriy Brun
brun@cs.umass.edu
University of Massachusetts Amherst
USA

ABSTRACT

Machine learning has radically changed what computing can accomplish, including the limits of what software engineering can do. I will discuss recent software engineering advances machine learning has enabled, from automatically repairing software bugs to data-driven software systems that automatically learn to make decisions. Unfortunately, with the promises of these new technologies come serious perils. For example, automatically generated program patches can break as much functionality as they repair. And self-learning, data-driven software can make decisions that result in unintended consequences, including unsafe, racist, or sexist behavior. But to build solutions to these shortcomings we may need to look no further than machine learning itself. I will introduce multiple ways machine learning can help verify software properties, leading to higher-quality systems.

CCS CONCEPTS

• **Software and its engineering**;

KEYWORDS

Machine learning and software engineering

ACM Reference Format:
Yuriy Brun. 2022. The Promise and Perils of Using Machine Learning When Engineering Software (Keynote Paper). In *Proceedings of the 6th International Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE '22), November 18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3549034.3570200

1 INTRODUCTION

Machine learning has significantly affected the software engineering process. For example, machine learning has been used for localizing faults [30, 55], automatically repairing bugs [32], requirement ambiguity detection and traceability [53], test generation [54], and bias enforcement [49], just to name a few applications. The promise of machine learning applications in software engineering is significant.

This keynote, however, examines some of the perils of using machine learning in the process of engineering software systems, and in the software systems themselves, and, then, discusses machine-learning-driven solutions to these perils. In particular, Section 2 examines pitfalls that can occur when automatically repairing programs, but Section 3 demonstrates how some of the same automated approaches can be applied to formal verification to avoid those exact pitfalls; and Section 4 illustrates how data-driven software that uses machine learning can result in unsafe or unfair behavior, but Section 5 summarizes how machine learning can help probabilistically verify safety and fairness properties to ensure systems are well-behaved.

MaLTeSQuE '22, November 18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9456-7/22/11.
https://doi.org/10.1145/3549034.3570200

2 PROGRAM REPAIR

Automated program repair attempts to reduce the cost of fixing bugs by automatically producing patches [17, 20] for software. The central idea is, when one or more tests fail because of a bug, fixing the bug often consists of editing the source code slightly to make all tests pass. This, in its essence, is a search problem, and computers are great at performing this kind of search. This search can be bounded by (1) using fault localization to identify the code locations most likely to be responsible for the bug, and (2) determining which code changes are worth attempting. Both of those processes can be supported by machine learning [30, 32, 55]. APR tools' success is best exemplified by their recent use in industry [7, 26, 33, 41].

Unfortunately, repair tools patch only a small fraction of defects correctly [38, 41, 43] and industrial deployments require significant manual oversight. For example, for Java, evaluated on the Defects4J benchmark [24], we have found that automated program repair techniques produce patches for 10–19% of the defects, and only 14–46% of those patches pass an independent set of tests [38]. This suggests that, while more than 80% of the time, automated program repair simply fails to produce a patch, worse, more than half of the time it does produce a patch it claims is correct, that patch either fails to repair the buggy functionality, or breaks other functionality in a way that existing test suites do not detect. So, while automated program repair exhibits significant promise, it can be overshadowed by the perils of producing patches that do more harm than good.

The fundamental cause of producing low-quality patches in program repair, known as overfitting, is that the function for deciding if a patch is correct — typically a test suite — is nearly always partial. And, therefore, as test suites necessarily undertest certain behavior, they allow for incorrect patches to appear correct. Increasing the granularity of the code changes has a marginal effect on improving the quality of repair [1, 25]. Similarly, attempts to improve fault localization have led to slight improvements in repair quality [37]. Applying program repair in some domains, such as build scripts, can lead to better quality [51] but is not a general solution. Recent work on generating oracles to improve test-suite quality [9, 13, 19, 36, 46] may potentially improve repair quality as well, though, today, it is limited in the kinds of behavior it can capture. The end result is,

unless we develop better methods for evaluating patch correctness, automated program repair is attempting to solve an underspecified problem and is doomed to never fully succeed.

3 AUTOMATED FORMAL VERIFICATION

Formal verification using interactive theorem provers, such as Coq [48] and Isabelle/HOL [40], is a promising method for building correct software. It has been used in industry, including by Airbus France, which uses the Coq-verified CompCert C compiler [28] to ensure safety and improve performance of its aircraft [44]. And Amazon successfully applies formal verification to cloud security problems in Amazon Web Services, providing tools for users to detect entire classes of misconfigurations that can potentially expose vulnerable data [6]. However, the manual effort involved in such formal verification is often prohibitive. For example, the Coq proof of the C compiler is more than three times that of the compiler code itself and took three person years of work [28]. Meanwhile, it took 11 person years to write the proof script to verify a microkernel [39]. As a general rule, because of the expense of verification, nearly all software companies ship is unverified.

But, unlike most programming languages, by their very nature, programs written in ones used for formal verification exhibit a special property — if a theorem prover says they are correct, they are guaranteed to be correct. This creates an ideal application of program repair and synthesis: formal verification program proofs are extremely effort intensive to write, but automatically generated ones, if a theorem prover agrees, are guaranteed to be correct.

Applying automated-program-repair technology to Coq proof script generation has been fairly successful. ASTactic can successfully, fully automatically prove 12.3% of the theorems in a large benchmark [52], while TacTok can prove 12.9% [15], and Diva can prove 21.7% [14]. These tools use machine-learning-based language modeling to learn a predictive model of a proof script, and then search through the space of possible proof scripts, guided by feedback from the theorem prover, to synthesize, from scratch, proof scripts. Recent advances, such as increasing the depth of information encoded in the models, shows even more promise [42].

4 DATA-DRIVEN SOFTWARE

Today, software that makes decisions is increasingly driven by machine learning, from online recommendation engines, to hiring decisions, to financial instrument availability, to decisions within our justice system. Machine learning has nothing short of revolutionized countless fields and applications, improving decision quality.

Unfortunately, there is amble evidence that the machine learning models can extract, and even exacerbate biases from its training data. These biases can show up in language modeling [12] and processing [10], automated transcription [47], facial recognition [27], ads [45], product and service availability [29], discount offers availability [21, 35], and criminal sentencing [5].

Thus, as the use of machine learning has enabled new applications and improved performance of data-driven systems, it has simultaneously created a new kind of software defect [11, 16], endangering the success of and creating a potential for harm software systems can cause.

5 PROBABILISTIC VERIFICATION OF MACHINE-LEARNING-BASED SOFTWARE

However, with the emergence of these new kinds of bugs that can result in unsafe or discriminatory behavior, new research has led to improvements in machine learning technology to provide guarantees of safety and fairness [49].

For example, FairSquare is able to provide probabilistic fairness guarantees for certain definitions of fairness for binary classifiers [3]. Meanwhile, machine learning models trained using the Seldonian framework [49] come with probabilistic guarantees that the model will not violate the user-specified safety or fairness properties, with high probability, on unseen data. For example, Seldonian algorithms can ensure that an update to an insulin pump causes no more instances of hypoglycemia than without the update, or that a model that recommends which candidates one should interview does not discriminate against gender and race [49].

Seldonian algorithms can be extended to work with contextual bandits to learn safe and fair policies [34]. The approach can also provide high-probability guarantees in settings when the training data and the data to which the model is applied in the field come from different distributions (even when only partial information about the in-field distribution is known) [18]. Finally, these ideas can be extended to definitions of delayed impact [31], that aim to enforce fairness in the long-term, rather than making seemingly fair decisions that are only fair in the short term but cause long-term harm [50].

Coupled with an emergence of tools that support fairness-aware decision-making in data-driven systems [2, 4, 8, 16, 22, 23], these verification methods can help produce not only machine learning models that are safe and fair, but also enable a better understanding of the safety and fairness requirements of software systems, and that better satisfy the needs of the users, both in the short term, and in the long-term future.

6 CONCLUSION

Machine learning is rapidly changing both software and the process of engineering that software. With these changes, many pitfalls emerge that may lead software to act in unexpected and undesirable ways. However, machine learning can enable not only methods to overcome these pitfalls, but also technology that will lead to higher quality software and engineering processes than ever before.

ACKNOWLEDGEMENT

This work is supported by the U.S. National Science Foundation under grants no. CCF-1763423 and CCF-2210243, and by the Defense Advanced Research Projects Agency (DARPA) under grant no. grant HR0011-22-9-0063.

REFERENCES

[1] Afsoon Afzal, Manish Motwani, Kathryn T. Stolee, Yuriy Brun, and Claire Le Goues. 2021. SOSRepair: Expressive Semantic Search for Real-World Program Repair. *IEEE Transactions on Software Engineering (TSE)* 47, 10 (October 2021), 2162–2181. https://doi.org/10.1109/TSE.2019.2944914

[2] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. 2018. A reductions approach to fair classification. In *International Conference on Machine Learning (ICML)*, Vol. PMLR 80. Stockholm, Sweden, 60–69.

[3] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya Nori. 2017. FairSquare: Probabilistic Verification for Program Fairness. In *ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*. Vancouver, BC, Canada.

[4] Rico Angell, Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2018. Themis: Automatically Testing Software for Discrimination. In *Proceedings of the Demonstrations Track at the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (6–9). Lake Buena Vista, FL, USA, 871–875. https://doi.org/10.1145/3236024.3264590

[5] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine Bias. *ProPublica* May 23 (2016). https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

[6] AWS 2022. AWS Provable Security. https://aws.amazon.com/security/provable-security.

[7] Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. 2019. Getafix: Learning to fix bugs automatically. *Proceedings of the ACM on Programming Languages (PACMPL) Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) issue* 3 (October 2019).

[8] Rachel K. E. Bellamy, Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, Seema Nagar, Karthikeyan Natesan Ramamurthy, John Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. 2018. AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias. *CoRR* 1810.01943 (2018). https://arxiv.org/abs/1810.01943

[9] Arianna Blasi, Alberto Goffi, Konstantin Kuznetsov, Alessandra Gorla, Michael D. Ernst, Mauro Pezzè, and Sergio Delgado Castellanos. 2018. Translating code comments to procedure specifications. In *International Symposium on Software Testing and Analysis (ISSTA)*. Amsterdam, Netherlands, 242–253. https://doi.org/10.1145/3213846.3213872

[10] Su Lin Blodgett and Brendan O'Connor. 2017. Racial Disparity in Natural Language Processing: A Case Study of Social Media African-American English. In *Fairness, Accountability, and Transparency in Machine Learning (FAT/ML)*. Halifax, NS, Canada.

[11] Yuriy Brun and Alexandra Meliou. 2018. Software Fairness. In *Proceedings of the New Ideas and Emerging Results Track at the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (6–9). Lake Buena Vista, FL, USA, 754–759. https://doi.org/10.1145/3236024.3264838

[12] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. *Science* 356, 6334 (2017), 183–186. https://doi.org/10.1126/science.aal4230

[13] Michael D. Ernst. 2017. Natural Language is a Programming Language: Applying Natural Language Processing to Software Development. In *Summit on Advances in Programming Languages (SNAPL)*, Vol. 71. Dagstuhl, Germany, 4:1–4:14. https://doi.org/10.4230/LIPIcs.SNAPL.2017.4

[14] Emily First and Yuriy Brun. 2022. Diversity-Driven Automated Formal Verification. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)* (22–27). Pittsburgh, PA, USA, 749–761.

[15] Emily First, Yuriy Brun, and Arjun Guha. 2020. TacTok: Semantics-Aware Proof Synthesis. *Proceedings of the ACM on Programming Languages (PACMPL) Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) issue* 4 (November 2020), 231:1–231:31. https://doi.org/10.1145/3428299

[16] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness Testing: Testing Software for Discrimination. In *Proceedings of the 11th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)* (6–8). Paderborn, Germany, 498–510. https://doi.org/10.1145/3106237.3106277

[17] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. 2019. Automatic Software Repair: A Survey. *IEEE Transactions on Software Engineering (TSE)* 45, 1 (2019), 34–67. https://doi.org/10.1109/TSE.2017.2755013

[18] Stephen Giguere, Blossom Metevier, Yuriy Brun, Bruno Castro da Silva, Philip S. Thomas, and Scott Niekum. 2022. Fairness Guarantees under Demographic Shift. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)* (25–29). https://openreview.net/forum?id=wbPObLm6ueA

[19] Alberto Goffi, Alessandra Gorla, Michael D. Ernst, and Mauro Pezzè. 2016. Automatic generation of oracles for exceptional behaviors. In *International Symposium on Software Testing and Analysis (ISSTA)*. Saarbrücken, Genmany, 213–224. https://doi.org/10.1145/2931037.2931061

[20] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated Program Repair. *Commun. ACM* 62, 12 (Nov. 2019), 56–65. https://doi.org/10.1145/3318162

[21] Devindra Haweawar. 2012. Staples, Home Depot, and other online stores change prices based on your location. *VentureBeat* December 24 (2012). https://venturebeat.com/2012/12/24/staples-online-stores-price-changes.

[22] Brittany Johnson, Jesse Bartola, Rico Angell, Katherine Keith, Sam Witty, Stephen J. Giguere, and Yuriy Brun. 2020. Fairkit, Fairkit, on the Wall, Who's the Fairest of Them All? Supporting Data Scientists in Training Fair Models. *CoRR* abs/2012.09951 (2020). https://arxiv.org/abs/2012.09951.

[23] Brittany Johnson and Yuriy Brun. 2022. Fairkit-learn: A Fairness Evaluation and Comparison Toolkit. In *Proceedings of the Demonstrations Track at the 44th International Conference on Software Engineering (ICSE)* (22–27). Pittsburgh, PA, USA, 70–74. https://doi.org/10.1145/3510454.3516830

[24] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. San Jose, CA, USA, 437–440. https://doi.org/10.1145/2610384.2628055

[25] Yalin Ke, Kathryn T. Stolee, Claire Le Goues, and Yuriy Brun. 2015. Repairing Programs with Semantic Code Search. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (9–13). Lincoln, NE, USA, 295–306. https://doi.org/10.1109/ASE.2015.60 DOI: 10.1109/ASE.2015.60.

[26] Serkan Kirbas, Etienne Windels, Olayori McBello, Kevin Kells, Matthew Pagano, Rafal Szalanski, Vesna Nowack, Emily Rowan Winter, Steve Counsell, David Bowes, Tracy Hall, Saemundur Haraldsson, and John Woodward. 2021. On The Introduction of Automatic Program Repair in Bloomberg. *IEEE Software* 38, 4 (2021), 43–51. https://doi.org/10.1109/MS.2021.3071086

[27] Brendan F. Klare, Mark J. Burge, Joshua C. Klontz, Richard W. Vorder Bruegge, and Anil K. Jain. 2012. Face Recognition Performance: Role of Demographic Information. *IEEE Transactions on Information Forensics and Security (TIFS)* 7, 6 (December 2012), 1789–1801. https://doi.org/10.1109/TIFS.2012.2214212

[28] Xavier Leroy. 2009. Formal verification of a realistic compiler. *Communications of the ACM (CACM)* 52, 7 (2009), 107–115. https://doi.org/10.1145/1538788.1538814

[29] Rafi Letzter. 2016. Amazon just showed us that 'unbiased' algorithms can be inadvertently racist. *TECH Insider* April 21 (2016). http://www.techinsider.io/how-algorithms-can-be-racist-2016-4.

[30] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. DeepFL: Integrating Multiple Fault Diagnosis Dimensions for Deep Fault Localization. In *International Symposium on Software Testing and Analysis (ISSTA)*. Beijing, China, 169–180. https://doi.org/10.1145/3293882.3330574

[31] Lydia T. Liu, Sarah Dean, Esther Rolf, Max Simchowitz, and Moritz Hardt. 2018. Delayed Impact of Fair Machine Learning. In *International Conference on Machine Learning (ICML)*, Vol. 80. PMLR 3150–3158.

[32] Fan Long and Martin Rinard. 2016. Automatic Patch Generation by Learning Correct Code. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. St. Petersburg, FL, USA, 298–312. https://doi.org/10.1145/2837614.2837617

[33] Alexandru Marginean, Johannes Bader, Satish Chandra, Mark Harman, Yue Jia, Ke Mao, Alexander Mols, and Andrew Scott. 2019. SapFix: Automated End-to-End Repair at Scale. In *ACM/IEEE International Conference on Software Engineering (ICSE)* (29–31). Montreal, QC, Canada.

[34] Blossom Metevier, Stephen Giguere, Sarah Brockman, Ari Kobren, Yuriy Brun, Emma Brunskill, and Philip S. Thomas. 2019. Offline Contextual Bandits with High Probability Fairness Guarantees. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS), Advances in Neural Information Processing Systems 32* (9–14). Vancouver, BC, Canada, 14893–14904. http://papers.neurips.cc/paper/9630-offline-contextual-bandits-with-high-probability-fairness-guarantees

[35] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. 2012. Detecting Price and Search Discrimination on the Internet. In *ACM Workshop on Hot Topics in Networks (HotNets)*. Redmond, Washington, 79–84. https://doi.org/10.1145/2390231.2390245

[36] Manish Motwani and Yuriy Brun. 2019. Automatically Generating Precise Oracles from Structured Natural Language Specifications. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)* (29–31). Montreal, QC, Canada, 188–199. https://doi.org/10.1109/ICSE.2019.00035

[37] Manish Motwani and Yuriy Brun. 2020. Automatically Repairing Programs Using Both Tests and Bug Reports. *CoRR* abs/2011.08340 (2020). https://arxiv.org/abs/2011.08340.

[38] Manish Motwani, Mauricio Soto, Yuriy Brun, René Just, and Claire Le Goues. 2022. Quality of Automated Program Repair on Real-World Defects. *IEEE Transactions on Software Engineering (TSE)* 48, 2 (February 2022), 637–661. https://doi.org/10.1109/TSE.2020.2998785

[39] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. 2013. seL4: From general purpose to a proof of information flow enforcement. In *IEEE Symposium on Security and Privacy (S&P)*. San Francisco, CA, USA, 415–429.

[40] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. 2002. *Isabelle/HOL: A proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media.

[41] Kunihiro Noda, Yusuke Nemoto, Keisuke Hotta, Hideo Tanida, and Shinji Kikuchi. 2020. Experience Report: How Effective is Automated Program Repair for Industrial Software?. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 612–616.

[42] Alex Sanchez-Stern, Emily First, Timothy Zhou, Zhanna Kaufman, Yuriy Brun, and Talia Ringer. 2022. Passport: Improving Automated Formal Verification Using Identifiers. *CoRR* abs/2204.10370 (2022). https://arxiv.org/abs/2204.10370.

[43] Edward K. Smith, Earl Barr, Claire Le Goues, and Yuriy Brun. 2015. Is the Cure Worse than the Disease? Overfitting in Automated Program Repair. In *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)* (2–4). Bergamo, Italy, 532–543. https://doi.org/10.1145/2786805.2786825 Previous versions appeared as University of Massachusetts Computer Science technical report UM-CS-2015-007 and as UC Davis College of Engineering technical report https://escholarship.org/uc/item/3z8926ks. DOI: 10.1145/2786805.2786825.

[44] Jean Souyris. 2014. Industrial Use of CompCert on a Safety-Critical Software Product. http://projects.laas.fr/IFSE/FMF/J3/slides/P05_Jean_Souyiris.pdf.

[45] Latanya Sweeney. 2013. Discrimination in Online Ad Delivery. *Communications of the ACM (CACM)* 56, 5 (May 2013), 44–54. https://doi.org/10.1145/2447976.2447990

[46] Shin Hwei Tan, Darko Marinov, Lin Tan, and Gary T. Leavens. 2012. @tComment: Testing Javadoc comments to detect comment-code inconsistencies. In *International Conference on Software Testing, Verification, and Validation (ICST)*. Montreal, QC, Canada, 260–269. https://doi.org/10.1109/ICST.2012.106

[47] Rachael Tatman. 2017. Gender and Dialect Bias in YouTube's Automatic Captions. In *Workshop on Ethics in Natural Language Processing*. Valencia, Spain. https://doi.org/10.18653/v1/W17-1606

[48] The Coq Development Team. 2017. Coq, v.8.7. https://coq.inria.fr.

[49] Philip S. Thomas, Bruno Castro da Silva, Andrew G. Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill. 2019. Preventing Undesirable Behavior of Intelligent Machines. *Science* 366, 6468 (22 November 2019), 999–1004. https://doi.org/10.1126/science.aag3311

[50] Aline Weber, Blossom Metevier, Yuriy Brun, Philip S. Thomas, and Bruno Castro da Silva. 2022. Enforcing Delayed-Impact Fairness Guarantees. *CoRR* abs/2208.11744 (2022). https://arxiv.org/abs/2208.11744.

[51] Aaron Weiss, Arjun Guha, and Yuriy Brun. 2017. Tortoise: Interactive System Configuration Repair. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (31–2). Urbana-Champaign, IL, USA, 625–636. https://doi.org/10.1109/ASE.2017.8115673

[52] Kaiyu Yang and Jia Deng. 2019. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*. Long Beach, CA, USA. http://proceedings.mlr.press/v97/yang19a/yang19a.pdf

[53] Kareshna Zamani, Didar Zowghi, and Chetan Arora. 2021. Machine Learning in Requirements Engineering: A Mapping Study. In *International Requirements Engineering Conference Workshops*. 116–125. https://doi.org/10.1109/REW53955.2021.00023

[54] Yixue Zhao, Saghar Talebipour, Kesina Baral, Hyojae Park, Leon Yee, Safwat Ali Khan, Yuriy Brun, Nenad Medvidovic, and Kevin Moran. 2022. AVGUST: Automating Usage-Based Test Generation from Videos of App Executions. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (14–18). Singapore. https://doi.org/10.1145/3540250.3549134

[55] Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D Ernst, and Lu Zhang. 2019. An Empirical Study of Fault Localization Families and Their Combinations. *IEEE Transactions on Software Engineering (TSE)* (2019). https://doi.org/10.1109/TSE.2019.2892102

Received 2022-07-22; accepted 2022-08-22