Keynote Talk: Adversarial Benchmarking

An Urgent Call for a Stronger Signal of Technological Progress

Marcel Böhme
marcel.boehme@acm.org
MPI for Security and Privacy
Germany

***Abstract*—Benchmarking has been our yardstick for evaluating technological progress. Today, we publish new coding agents, new testing techniques, and generally new tools for important problems enthusiastically with strong results on the most recent benchmarks. These results are offered as empirical evidence in support of our claims of superiority: "This is the new state-of-the-art". These days, the best-performing LLMs and agentic frameworks go viral on a regular basis. However, recently there has been increasing skepticism of benchmarking as our primary methodology for evaluating technological progress: Overfitting. Results over insight. Unreliable measures of effectiveness. Expectation bias in the peer review process. An entire community misled by an invalid problem statement. Do we need better benchmarks? Or do we need a fundamentally different approach?**

In this keynote, I argue that we must—with the same enthusiasm—explicitly identify and evaluate the unique limitations of the techniques we develop. How else do we learn where to improve? In fact, this is precisely the perspective that the Software Testing research community takes on its own subject: We want to find bugs! We don't want to confirm that a system is effectively achieving its purpose. That would be the least efficient way to establish where more progress is needed. Benchmarking is like providing a test suite with a Protocol RFC that is supposed to find bugs in every future protocol implementation. To evaluate new technology properly, we must stop seeking to confirm its effectiveness and start failing to find counterexamples. This is our call to develop the scientific standards for evaluating technological progress for all of computer science. We must set new expectations for peer review to allow, invite, and indeed request a fair analysis of the unique limitations of the techniques we propose. I will provide several concrete examples and mechanisms for how we can facilitate this "adversarial benchmarking" and call on the Software Testing research community to develop these ideas so as to ultimately facilitate a sound technological progress.

I. Necessity of Benchmarking

How can we evaluate the scientific progress in an area of research? How can the layperson distinguish an effective from an ineffective technological suggestion for improvement? How do *you* know whether a scientific claim is actually true?

Benchmarking measures progress. Benchmarking is our primary means of evaluating technological progress. It offers *empirical evidence* to a paper's claims that a newly proposed approach to solve a problem P improves over the state-of-the-art (i.e., existing approaches to solve P) or over the baseline approach that was extended. Empirical evidence gives meaning to such claims and inspires confidence in the non-expert. In a 2019 Shonan meeting on Fuzzing and Symbolic Execution with a balanced participation from both industry and research, benchmarking was considered one of the Top-3 most important research challenges going forward [1].

To measure how well our new approach A solves a problem P, we first collect several representative *problem instances* $\mathcal{B}$ and define a *measure of performance* M. We implement our approach A into tool T_A by extending a tool T_0. For instance, in automated software testing, we may implement a new power schedule A into a tool `AFL3+` (T_A) by extending `AFL++` [2] (T_0) and compare both in terms of the coverage achieved (M) on the programs ($\mathcal{B}$) in FuzzTastic [3]. If T_A performs better on B in terms of M than T_0, we consider A as *effective*. If T_A performs better on B in terms of M than the implementation of any previous state-of-the-art (SOTA) approach, we consider our approach A as the *new SOTA*.

Benchmarking induces progress. Substantial improvements over the recent SOTA on the most popular benchmarks are widely expected when new LLMs or LLM-based agentic frameworks are introduced. There are extremely popular leaderboards [4], [5] and dedicated competition tracks at top conferences. There are benchmarking competitions that provide huge financial incentives to stimulate new advances on important problems (e.g., the ARC Prize offers 2 million USD; AIxCC offered 8.5 million USD in prizes). Benchmarking and the facilitating regular or long-running competitions have become effective drivers for innovation.

Problem of induction & statistical analysis. Hume argued that we can never *confirm* a scientific theory ("all swans are white") just by collecting more evidence in favor. Similarly, we can never absolutely confirm a claim that T_A performs better than T_0 in terms of M just by collecting more evidence in favor. However, if we assume that our benchmark $\mathcal{B}$ is a random sample from an (unknown) "real world" distribution of problem instances, we can assign a probability to the validity of the claim. By running a statistical analysis, we can quantify the effect size and the statistical significance of the difference in performance [6]–[8]. For a textbook on the emerging science of benchmarking with a focus on machine learning, we refer to Hardt [9]. Under the random sample assumption, we actually know that the proportion of problem instances that are of a kind that is not represented in $\mathcal{B}$ is surprisingly almost entirely determined by $\mathcal{B}$ [10].

Nevertheless, Hume's problem of induction continues to hold. With a focus on seeking to confirm the effectiveness of an approach, we ultimately consider as effective too many approaches that are actually ineffective. Multiple, potentially even contradictory approaches can explain the same empirical evidence. This is Popper's problem of under-determination.

979-8-3195-3308-1/26/$31.00 ©2026 IEEE
DOI 10.1109/ICST69053.2026.00097

Without critically testing the limitations of superiority claims for new technology, we deprive ourselves of the opportunity to understand whether the observed progress is real, where we failed to make progress, and where more progress is needed.

Benchmarking is necessary. In the Software Engineering research community we now often expect the empirical evidence of a paper's claims already at submission time. Authors would submit a link to a paper's artifacts together with the paper, and reviewers would do a lightweight check if the empirical evidence that can reasonably be expected is indeed provided. Moreover, a separate Artifact Evaluation Committee would help improve reproducibility and indicate artifact availability and reproducibility directly on the published paper.

II. Insufficiency of Benchmarking

There are several reasons why substantial improvements on a benchmark $\mathcal{B}$ in terms of a performance measure M do not always translate into technological progress on the problem P.

Overfitting. Once a tool is set up for continuous evaluation in a benchmarking framework, it is difficult to prevent overfitting to that specific benchmark. Once the maintainer of a tool T_A starts evaluating every potential improvement against the same benchmark $\mathcal{B}$, the evaluations become dependent, in a statistical sense. The performance of T_A on $\mathcal{B}$ may increase while the performance of T_A on a separate (hidden) benchmark $\mathcal{B}'$ (that is constructed using the same selection method) may remain the same or even decline. The observed performance increase is specific to the benchmark. In machine learning, we can design a reusable holdout [11] (i.e., the dataset that is held out for evaluation) to mitigate overfitting.

Results over insight. Achieving strong results in support of a claim is a means to an end. Our objective in this case is to have confidence in a claim of effectiveness or the validity of a hypothesis. However, achieving strong results has become an end in itself. As Goodhart put it: "When a measure becomes a target, it ceases to be a good measure" [12]. This focus on results induces an expectation bias, both in the reviewer and the author [13], [14]. It leaves little space to develop a deeper understanding also of the limitations of various approaches to solving a problem and fundamentally inhibits theory building.

If there is an incentive for both reviewers and authors to prefer strong positive results over the underlying theory and claims they are meant to support, then perhaps we ought to change the publication process itself. To this end, preregistration offers a two-stage publication process.[1] In *Stage 1*, the authors submit a full paper without the results, i.e., before the experiments are actually conducted. The reviewers evaluate the significance and novelty of the technique, and the soundness and reproducibility of the methodology specified to validate the core claims. If the registered report is accepted, the authors are invited to conduct the experiments and submit Stage 2. The paper is considered "in-principle accepted". In *Stage 2*, the reviewers only evaluate the degree to which the proposed study design has been followed, and the degree to which any deviations have been explained. Preregistration encourages the exploration of risky ideas and asks reviewers to focus on the novelty and significance of the idea, rather than the results.

Unreliable measures of performance. We quantify the degree to which a new approach improves on the SOTA or the baseline using a concrete, operational measure of performance M. For instance, in automated fault localization, we often make various assumptions, perhaps as a way to automate what would naturally deserve a user study [15]. In automated software testing, since software bugs are naturally very rare, we often measure a testing tool's bug-finding performance by the code coverage it achieves, but is coverage really a good measure of bug finding? My team and collaborators from Google found that a ranking of fuzzers $\mathcal{T} = \{T_{A1}, \cdots, T_{An}\}$ in terms of coverage M_{Cov} may not strongly agree with a ranking of $\mathcal{T}$ in terms of the number of bugs found M_{Bug} [8]: The best fuzzer in terms of coverage may actually be the worst in terms of bug finding. A collaborator from Meta conjectured that this result may be understood as a consequence of M_{Bug} being a rather noisy measure of performance. Indeed, we found that a ranking of $\mathcal{T}$ in terms of coverage M_{Cov} on a benchmark set $\mathcal{B}_1$ often agrees more with the ranking of $\mathcal{T}$ on a different, equi-sized benchmark set $\mathcal{B}_2$ in terms of M_{Bug} than a ranking on $\mathcal{B}_1$ in terms of M_{Bug} [16]. In other words, bug-based benchmarking may be so noisy that it is a worse predictor of an independent bug-based benchmarking outcome than coverage-based benchmarking.

This demonstrates that we cannot assume that our measures of performance represent technological progress faithfully. This is an instance of the Duham-Quine problem: One cannot test a scientific hypothesis in isolation because any test relies on a network of background assumptions. Hence, we should carefully test the limits of our operational measures of performance M and the assumptions we make to understand the degree to which an observed performance increase actually represents technological progress.

Invalid problem statements. Even a faithful performance measure is useless if the problem statement itself is invalid. We recently studied why machine learning for vulnerability detection (ML4VD) report unusually high values for precision and recall compared to the tools used in industry [17], [18]. ML4VD is most often defined as a binary classification problem: "Given a function, is it vulnerable?" We call a function vulnerable if the program was assigned a CVE and fixing this function removes the vulnerability. We studied 100 randomly sampled functions labeled as vulnerable in each of the 3 studied datasets [18]. We found that we cannot decide whether a function causes the program to be vulnerable without further context. Not as a security researcher. Not as a machine learning classifier. For completeness, we also looked at functions that are labeled as *not* vulnerable and found in most cases, we could create a context in which this function would be vulnerable for the same reasons. This means there is no information in the function that would explain the vulnerability label. The problem statement itself is ill-defined.

[1] The process described subsequently, I have helped to implement at ACM TOSEM (https://dl.acm.org/journal/tosem/registered-papers) and has been adopted at ACM TOPLAS (https://dl.acm.org/journal/toplas/pl-experiments).

Map of effectiveness. Using a measure M induces a single *axis* along which technological progress is measured and assumes that everyone has the same distribution over the instances of a problem. More realistically, we should think of a *map* that describes precisely under which circumstances a claim of superiority holds. In fact, it is impossible for a tool to outperform all other tools on all instances of a problem (No Free Lunch) [19]. We recently proposed a methodology to measure the degree to which the benchmarking outcome depends on the specific benchmark properties [20]. This allows us to report the benchmarking outcome *counter-factually*. For instance, we found: If FuzzBench had smaller programs, LibFuzzer would outperform all other fuzzers.

III. Adversarial Benchmarking and Critical Rationalism

Popper was convinced that seeking to confirm a scientific hypothesis will keep too many incorrect hypotheses in tact. Instead, he proposed, we should subject a hypothesis to critical testing and only failing to find counter-examples, tentatively consider the hypothesis as accepted. This is the objective of the scientific method and the subject of critical rationalism. Similarly, I argue that we must explicitly identify and evaluate the unique limitations of the approaches we develop. How else do we learn where to improve? In fact, this is precisely the perspective the Software Testing research community takes on its own subject: We want to find bugs! We don't want to confirm that a system achieves its purpose. That would be the least efficient way to establish where more progress is needed.

Time for change. Benchmarking is like providing a test suite of expected behaviors together with the specification of a protocol (i.e., a request for comments; RFC), expecting the test suite to find bugs in every future protocol implementation. To evaluate new technology properly, we must stop seeking to confirm its effectiveness and start failing to find counterexamples [21]. Instead of designing benchmarks to confirm expected performance, we should deliberately stress edge cases: worst-case inputs, pathological scenarios, and adversarial workloads. Instead of asking "Does my approach perform well?" or "Which approach is best", we should ask "Where does an approach *stop* performing well?". Instead of evaluating an approach along a single performance axis, we should build a "map of effectiveness" to chart out, in the problem space, *under which conditions* various approaches perform best [20]. Instead of expecting empirical evidence in favor of a claim *a priori*, we should expect *and provide the required space* for an honest discussion of the limits and challenges of newly proposed approaches (e.g., using preregistration). Instead of ignorance, when unreservedly accepting a substantial improvement on a popular benchmark, we should employ critical rationalism recursively and systematically study all components of our evaluation methodologies, including our assumptions [15], our measures of performance [8], [16], [17], and the validity of our problem statements to begin with [18].

Timeliness. With the advent of automatic artifact evaluation [22] and agentic coding [23], this is the right time to *automate* my proposal of adversarial benchmarking. Andreas Zeller, in his 2026 Harlan D. Mills acceptance speech, conceptually explored the opportunities of *auto-experimentation* when developing new approaches which seems to be within reach.

Call to arms. Software Engineering, Machine Learning and other research communities that are concerned with automated methods are faced with increasing skepticism of benchmarking as their primary methodology for evaluating technological progress: Overfitting. Results over insight. Unreliable measures of effectiveness. Expectation bias in the peer review process. An entire community misled by an invalid problem statement. To address this skepticism, we must set new expectations for peer review to allow, invite, and indeed request a fair analysis of the unique limitations of the approaches we propose. I would argue that it is the responsibility of a research community to *critically* test the claims that it makes, and that it is an opportunity for the software testing community to drive this research agenda: In addition to automated benchmarking which facilitates the confirmation of claims of effectiveness, we should develop automated techniques to try and *falsify* the primary claims in a paper—if only to understand the limits of an approach and opportunities to improve it. We should develop and automate this concept of "adversarial benchmarking" so as to ultimately facilitate sound technological progress across all of computer science.

Acknowledgment

I would like to express my gratitude to the Chairs of the ICST Program Committee Neil Walkinshaw and Xiaoyuan Xie and the General Chairs Moonzoo Kim and Shin Hong for their kind invitation, as well as their organizing committee for their wonderful support and organization. Your invitation to deliver a keynote talk at ICST, a top conference for software testing, at such an important turning point not only for the scientific research community as a whole but also for me personally, has been an incredible honor.

References

[1] M. Böhme, C. Cadar, and A. Roychoudhury, "Fuzzing: Challenges and reflections," *IEEE Software*, vol. 38, no. 3, pp. 79–86, 2021.

[2] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, "Afl++: combining incremental steps of fuzzing research," in *Proceedings of the 14th USENIX Conference on Offensive Technologies*, ser. WOOT'20. USA: USENIX Association, 2020.

[3] S. Lipp, D. Elsner, T. Hutzelmann, S. Banescu, A. Pretschner, and M. Böhme, "Fuzztastic: a fine-grained, fuzzer-agnostic coverage analyzer," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 75–79. [Online]. Available: https://doi.org/10.1145/3510454.3516847

[4] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, "SWE-bench: Can language models resolve real-world github issues?" in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=VTF8yNQM66

[5] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," 2021. [Online]. Available: https://arxiv.org/abs/2009.03300

[6] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1–10. [Online]. Available: https://doi.org/10.1145/1985793.1985795

[7] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating fuzz testing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2123–2138. [Online]. Available: https://doi.org/10.1145/3243734.3243804

[8] M. Böhme, L. Szekeres, and J. Metzman, "On the reliability of coverage-based fuzzer benchmarking," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22, 2022, pp. 1–13.

[9] M. Hardt, *The Emerging Science of Machine Learning Benchmarks*. Princeton University Press, 2026.

[10] S. Lee and M. Böhme, "How much is unseen depends chiefly on information about the seen," in *Proceedings of the 13th International Conference on Learning Representations*, ser. ICLR'25, 2025.

[11] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth, "The reusable holdout: Preserving validity in adaptive data analysis," *Science*, vol. 349, no. 6248, pp. 636–638, Aug. 2015.

[12] C. A. E. Goodhart, *Problems of Monetary Management: The UK Experience*. London: Macmillan Education UK, 1984, pp. 91–121. [Online]. Available: https://doi.org/10.1007/978-1-349-17295-5_4

[13] A. Zeller. (2019) When results are all that matters: The case of the angora fuzzer. [Online]. Available: https://andreas-zeller.info/2019/10/17/when-results-are-all-that-matters.html

[14] ——. (2019) When results are all that matters: Consequences. [Online]. Available: https://andreas-zeller.info/2019/10/10/when-results-are-all-that-matters-case.html

[15] E. Soremekun, L. Kirschner, M. Böhme, and M. Papadakis, "Evaluating the impact of experimental assumptions in automated fault localization," in *Proceedings of the ACM/IEEE 45th International Conference on Software Engineering*, ser. ICSE 2023, 2023, pp. 1–13.

[16] A. Madadi, S. Lee, C. Aschermann, and M. Böhme, "In bugs we trust? on measuring the randomness of a fuzzer benchmarking outcome," in *Proceedings of the ACM International Conference on the Foundations of Software Engineering*, ser. FSE'26, 2026.

[17] N. Risse and M. Böhme, "Uncovering the limits of machine learning for automatic vulnerability detection," in *Proceedings of the 33rd USENIX Security Symposium*, ser. USENIX Sec'24, 2024.

[18] N. Risse, J. Liu, and M. Böhme, "Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection," in *Proceedings of the 34th ACM/SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA, 2025.

[19] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Trans. Evol. Comp*, vol. 1, no. 1, p. 67–82, Apr. 1997. [Online]. Available: https://doi.org/10.1109/4235.585893

[20] D. Wolff, M. Böhme, and A. Roychoudhury, "Fuzzing: On benchmarking outcome as a function of benchmark properties," *ACM Transactions on Software Engineering and Methodology*, 2025.

[21] M. Böhme, "How to solve cybersecurity once and for all," *IEEE Security and Privacy*, vol. 23, 2025.

[22] D. Baek and M. Pradel, "Artisan: Agentic artifact evaluation," 2026. [Online]. Available: https://arxiv.org/abs/2602.10046

[23] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury, "Autocoderover: Autonomous program improvement," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 1592–1604. [Online]. Available: https://doi.org/10.1145/3650212.3680384