Using Benchmarking Bots for Continuous Performance Assessment

Florian Markusse*, Philipp Leitner†, Alexander Serebrenik*

* Eindhoven University of Technology
Eindhoven, The Netherlands
f.j.markusse@student.tue.nl, a.serebrenik@tue.nl

† Chalmers University of Technology
Gothenburg, Sweden
philipp.leitner@chalmers.se

***Abstract*—Bots for continuous performance assessment, in short benchmarking bots, are starting to see use as a productivity tool, helping large open source projects judge whether new code contributions negatively impact performance. We discuss how and why projects use benchmarking bots, and present an in-depth case study of The Nanosoldier bot used by the team behind the Julia programming language.**

Introduction

Software development is a time-consuming and labor-intensive activity. Hence, it's not surprising that bots are already widely spread, both in industry and in the open source ecosystem. Bots handle routine tasks ranging from updating dependencies to continuous quality assessment, e.g., by running test suites or static code analysis tools [1].

However, so far, much less attention has been paid to the usage of bots for continuous performance assessment, even though software performance is a critical quality concern for many projects. This is understandable given that even experienced developers often struggle to correctly benchmark their systems, or judge whether an observed slowdown is more likely to be statistical noise or an actual performance regression [2]. Still some projects have taken on the challenge to build bespoke automated performance benchmarking bots.

A brief summary of our research methods and the data sources can be found in Sidebar 1.

Sidebar 1: Data Sources and Analysis

The findings described in this article are based on the research conducted by the first author during his master's project [3]. The overall goal of our research is to construct a theory of the usage of a benchmarking bots. In this paper, we focus on an empirical characterization of benchmarking bot usage in open source projects. We have comprehensively investigated multiple well-known projects: the Julia, Apple Swift, and Rust programming languages, the Diem decentralized database, and Microsoft's Fluent UI web framework. All of these projects have adopted a benchmarking bot as part of their code review process. This dataset can be found on Zenodo (bit.ly/3D2QcNM).

We have quantitatively compared pull requests (PRs) where the benchmarking bot was involved with PRs where it was not, with regards to the number of reviews, number of commits, amount of discussion, and number of developers involved in the discussion.

Additionally, we performed an in-depth study of the Julia project, a large and mature project that has adopted a bot in April 2016. We have interviewed two developers about their experiences with The Nanosoldier, the bot that the Julia project uses as well as asked them to interpret the results of quantitative comparison of PRs.

Benchmarking Bots

Benchmarking bots are not yet a common feature of open source software development. Using keywords stated in Appendix A [3] we search GitHub for pull requests referring to performance regression or improvement and manually checking whether the pull requests were related to bots for continuous performance assessment. We say that a project has adopted the benchmarking bot if more than 10 pull requests in a year use this benchmarking bot. In this way we identified 11 projects that have adopted or built such a bot. Even though we do not claim this list to be complete, the number of projects using a benchmarking bot is surprisingly low, especially considering the ubiquitousness of other CI bots, such as those for dependency management [1]. Table I provides an overview of identified projects using a benchmarking bot.

Projects that adopt benchmarking bots tend to be large, and, unsurprisingly, in performance-sensitive domains, such as programming languages (Julia, Swift, Rust) or frameworks for Web development (Salesforce LWC, Hexo, or Microsoft's Fluent UI). We also observe that many projects in our list adopted their benchmarking bot as recently as 2020, with the notable exception of the Julia programming language, which has been using a benchmarking bot (The Nanosoldier) since early 2016. Finally, the benchmarking bots we observed in our study are all custom-built or heavily customized for usage in their projects. As of today, no "standard benchmarking bot" has emerged that sees usage across the open source ecosystem, akin to build or dependency management bots.

Project	Domain	Bot	Adoption	Trigger	Reporting
JuliaLang/julia	Prog. language	*The Nanosoldier*	Apr-16	Manual	PR Comment
apple/swift	Prog. language	*swift-ci*	Jan-17	Manual	PR Comment
guardian/frontend	Website	*PRBuilds*	Jun-17	Unknown*	PR Comment
rust-lang/rust	Prog. language	*rust-timer*	Aug-18	Manual	PR Comment
Salesforce/lwc	Web	*Salesforce Best*	Jul-19	Unknown*	PR Comment
diem/diem	Cryptocurrency	*github-actions*	Mar-20	Manual	PR Comment
PaddlePaddle/Paddle	Machine learning	*paddle-bot*	May-20	Automatic	PR Comment
hexojs/hexo	Web	*github-actions*	Jul-20	Automatic	Checkmark
simdjson/simdjson	Parser	*github-actions*	Sep-20	Automatic	Checkmark
ibis-project/ibis	Data analysis	*github-actions*	Oct-20	Automatic	Checkmark
microsoft/fluentui	Web	*fabricteam*	Dec-20	Automatic	PR Comment

*: No explicit invocation event can be detected; contributors likely trigger the benchmarking bot internally.

TABLE I: Overview of Projects Using Benchmarking Bots

Two interesting observations relate to how developers interact with benchmarking bots. Firstly, there are two different ways to surface benchmarking results to developers: most bots add comments that link to detailed reports to pull request discussion threads; only three projects (simdjson, hexo, and ibis) surface benchmark results as GitHub Actions "checkmarks". This is surprising given that similar badges are commonly used for other project quality checks [4].

Secondly, despite being integrated into the CI pipeline, four projects elect to not execute their benchmarking bots automatically and on every pull request. Instead, they have to be manually triggered by a developer, for instance by getting tagged in the code review discussion of a pull request. Given that system benchmarking is well-known to be time-consuming and resource-intensive [5], these projects are selective with how often they choose to run their benchmarking bots. For example, in the Julia project, only 5% of pull requests are actually benchmarked, and only 8% of contributors have ever interacted with the benchmarking bot.

An example of a developer invoking The Nanosoldier in Julia is given in Figure 1(a). Starting a benchmark requires tagging in the `@nanosoldier` user. Notably, no attempts are made to provide a particularly "conversational" interface. Instead, the benchmarking bot is called upon through a syntax clearly inspired by method invocation. After benchmarking is completed, the bot links to a report in the discussion thread. Further note that one specific developer is tagged by the benchmarking bot for every report.

The Nanosoldier and the Julia Project

To explore the impact that adopting a benchmarking bot for continuous performance assessment can have, we now explore the Julia project in more detail as a case study. We selected the Julia project as the focal point of this case study due to its importance as a state-of-the-art programming language in data science, the large size of its code base, and the maturity of its benchmarking bot usage.

Specifically, we retrieved all pull requests with at least two source file changes of the Julia project since 2016. We added the requirement for source file changes to exclude "simple" pull requests, such as documentation modifications. Contributors invoke The Nanosoldier in a subset of pull requests, and we try to isolate this group by this requirement.

Then, we grouped them into pull requests where the benchmarking bot was invoked at least once (N = 534) versus pull requests where this was not the case (N = 6957).

Figure 1(b) and Figure 1(c) quantitatively compares these two groups along two example dimensions: how many participants were involved in the discussion and how long the discussion was in terms of number of comments.

We observe that pull requests where The Nanosoldier gets invoked involve more participants and more comments. Additionally (not depicted for brevity), our research has also indicated that discussions involving the bot require more commits and lead to longer comment threads as measured by the total length of all comments taken together. The differences are statistically significant ($p < 0.0001$), with Cliff's δ effect sizes between small (0.28; number of commits) and large (0.5; number of participants). Hence, it appears that The Nanosoldier is predominately involved in the discussion of complex issues. However, based on this analysis alone it is unclear if the bot *causes* long, complex discussions, or if it is commonly brought into issues that simply are inherently more complex. To investigate this more deeply, we also conducted semi-structured interviews with two active and experienced Julia developers who have had ample interaction with The Nanosoldier.[1]

Both interviewees are convinced that The Nanosoldier is a beneficial tool in Julia's ecosystem, from its inception to the current day. Specifically, they relate The Nanosoldier to continuous integration, in that they only noticed how important the tool actually is when it was unavailable:

> *It's kind of like turning off CI. You think, oh, most things will stay mostly stable, what could go wrong [when it's offline]? And then you turn it back on, and there's like ten bugs.*

[1]This interview study has been approved by the Ethical Review Board of Eindhoven University of Technology, The Netherlands, ref. ERB2021MCS9.

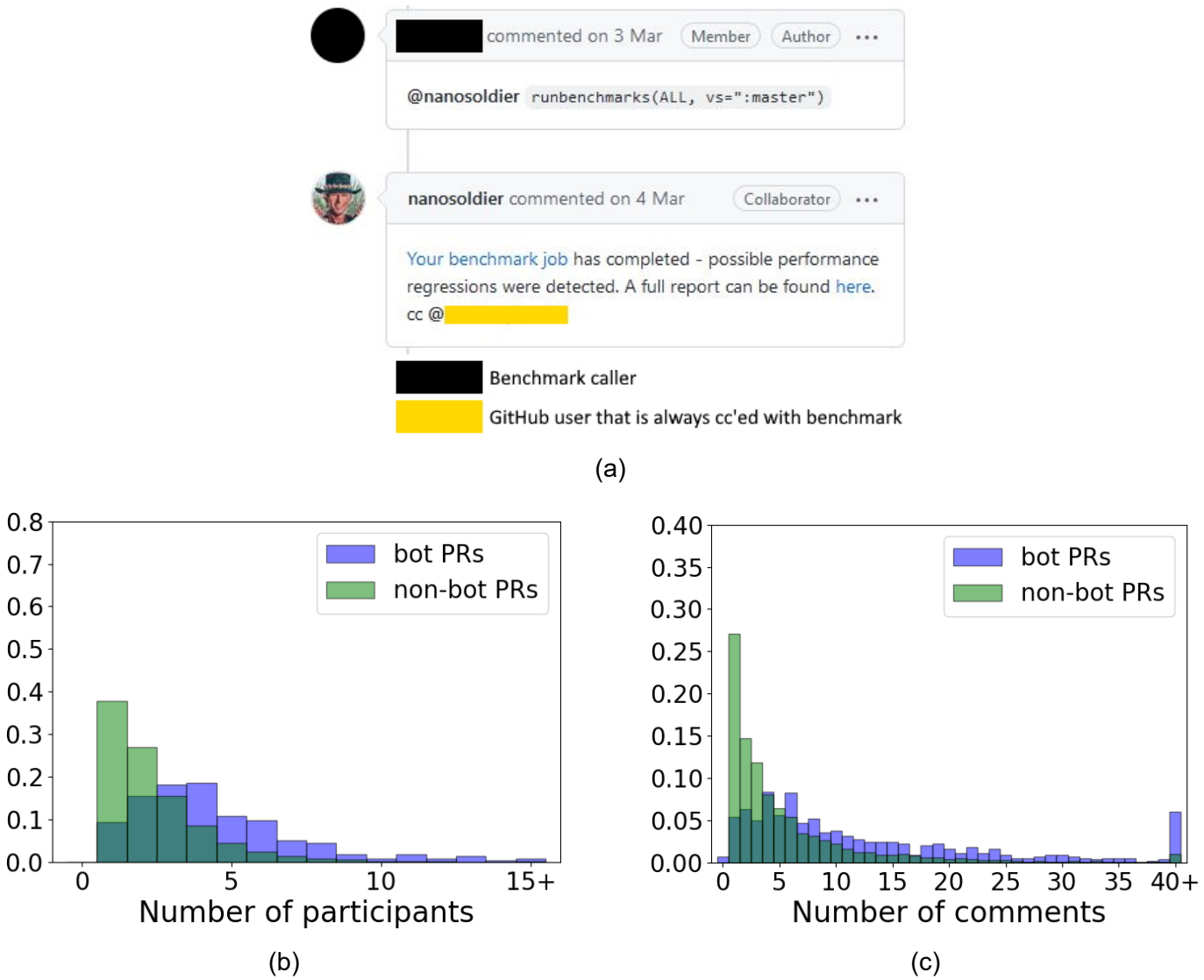

Fig. 1: (a) Invocation of The Nanosoldier in Julia (PR 39742); (b) Number of Unique Discussion Participants; (c) Number of Comments in Discussion. Dark green shaded areas in plots (b) and (c) indicate overlapping curves.

Our study indicates that developers appreciate the peace of mind that using a performance benchmarking bot gives them. Before merging a complex PR, they would often run The Nanosoldier to ensure the codebase did not unknowingly incur performance penalties. Furthermore, they confirmed our findings that PRs with a contribution from The Nanosoldier have a greater number of interactions, particularly because the report from the bot explicitly triggers discussion.

Furthermore, the question remains as to why The Nanosoldier is not invoked on every pull request. Indeed, having the bot profile a pull request takes a lot of time. Running the bot on each issue would overload, especially since benchmarks need to be run on controlled hardware to be meaningful:

> *Our benchmarks just take too long. If they were really fast, we would just run them on CI. But a combination of the fact that we're using wall clock time and that we need to run it on a machine that's closer to the actual hardware and less susceptible to interrupts.*

Thus, invoking The Nanosoldier is reserved for pull requests that are judged to be potentially high-impact and for which there are existing benchmarks. Additionally, the pull request often contains changes for which it is difficult to ascertain the performance impact:

> *... when just one little thing is changed and that could also be a PR that we run The Nanosoldier on because that little thing might be in a very crucial part of the source code. But usually, I would say that if The Nanosoldier is run on it [a PR], it is probably a bit more impactful.*

Sidebar 2: Other Research on Bots and Benchmarking

Bots in software development are an active research area as witnessed by a series of BotSE workshops, an eponymous Dagstuhl Seminar [6] and a current theme issue. The research so far has focused on understanding what kind of automation can be seen as a bot [7], [8] and how bots influence software development practices [9]. Another emerging research trend in the area is the automated identification of bot contributions [10], [11].

Performance of software systems has been extensively studied [12], [13]. Despite this the problem of performance evaluation in practice is far from being solved: Laaber & Leitner [14] observed big disparities between benchmarking suites in their ability to detect slowdowns, Costa et al. [2] found that 28% of the projects had at least a single instance of an incorrectly written benchmark, while Stefan et al. [15] observed that less than half a percent of the repositories used a performance benchmarking framework at all, let alone a benchmarking bot.

Outlook

Bots for (continuous) performance benchmarking are still rarely used in open-source software development. However, the introduction of GitHub Actions has made such bots more accessible to developers. We observe that four of eleven investigated benchmarking bots are built on top of GitHub Actions. Based on our research, we encourage developers of other performance-sensitive projects to consider adopting bot-based benchmarking—our results show significant positive impacts on projects such as Julia. Benchmarking bots allow contributors to test performance more readily while expending less effort concurrently. This could likely lead to more performance bugs being discovered earlier on in the development process. Furthermore, maintainers rest more easily knowing that complex changes either are automatically benchmarked, or that performance assessment can at least be triggered easily if necessary.

This being said, initial upfront investment is necessary. As of today, there is no "standardized benchmarking bot" (in the spirit of Dependabot and similar tools) that can be adopted easily, requiring interested projects to build largely bespoke benchmarking infrastructures and bots. However, with the advent of GitHub Actions this process has been greatly simplified. Still, we argue that there is currently a void to be filled regarding out-of-the-box re-usable benchmarking bots to bring this area into mainstream software development.

References

[1] M. Wessel, B. M. De Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa, "The power of bots: Characterizing and understanding bots in oss projects," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–19, 2018. 1

[2] D. E. D. Costa, C.-P. Bezemer, P. Leitner, and A. Andrzejak, "What's wrong with my benchmark results? studying bad practices in jmh benchmarks," *IEEE Transactions on Software Engineering*, 2019. 1, 4

[3] F. Markusse, "The impact of benchmarking bots on open-source software projects," Master's thesis, Eindhoven University of Technology, 2021, https://research.tue.nl/en/studentTheses/the-impact-of-benchmarking-bots-on-open-source-software-projects. 1

[4] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use github actions to automate their workflows?" in *International Conference on Mining Software Repositories*, 2021, pp. 420–431. 2

[5] V. Tarasov, S. Bhanage, E. Zadok, and M. I. Seltzer, "Benchmarking file system benchmarking: It *is* rocket science." in *HotOS*, vol. 13, 2011, pp. 1–5. 2

[6] M.-A. D. Storey, A. Serebrenik, C. P. Rosé, T. Zimmermann, and J. D. Herbsleb, "Botse: Bots in software engineering (dagstuhl seminar 19471)." Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. 4

[7] C. Lebeuf, A. Zagalsky, M. Foucault, and M.-A. D. Storey, "Defining and classifying software bots: a faceted taxonomy," in *BotSE@ICSE*, E. Shihab and S. Wagner, Eds. IEEE / ACM, 2019, pp. 1–6. 4

[8] L. Erlenhov, F. G. de Oliveira Neto, and P. Leitner, "An empirical study of bots in software development: characteristics and challenges from a practitioner's perspective," in *ESEC/FSE*. ACM, 2020, pp. 445–455. 4

[9] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa, "Effects of adopting code review bots on pull requests to oss projects," in *ICSME*. IEEE, 2020, pp. 1–11. 4

[10] T. Dey, S. Mousavi, E. Ponce, T. Fry, B. Vasilescu, A. Filippova, and A. Mockus, "Detecting and characterizing bots that commit code," in *Proceedings of the 17th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 209 – 219. [Online]. Available: https://doi.org/10.1145/3379597.3387478 4

[11] M. Golzadeh, A. Decan, D. Legay, and T. Mens, "A ground-truth dataset and classification model for detecting bots in github issue and pr comments," *Journal of Systems and Software*, vol. 175, p. 110911, 2021. 4

[12] M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *FOSE*. IEEE, 2007, pp. 171–187. 4

[13] T. Yu and M. Pradel, "Pinpointing and repairing performance bottlenecks in concurrent programs," *Empirical Software Engineering*, vol. 23, no. 5, pp. 3034–3071, 2018. 4

[14] C. Laaber and P. Leitner, "An evaluation of open-source software microbenchmark suites for continuous performance assessment," in *MSR*. IEEE, 2018, pp. 119–130. 4

[15] P. Stefan, V. Horky, L. Bulej, and P. Tuma, "Unit testing performance in java projects: Are we there yet?" in *ICPE*, 2017, pp. 401–412. 4