Social Proof is in the Pudding: The (Non)-Impact of Social Proof on Software Downloads

Lucas Shen, Gaurav Sood, and Daniel Weitzel

Abstract. Open-source software is widely used in commercial applications. Paired with the fact that developers often use social proof as a cue when choosing which open-source software to use, these two facts raise concerns that bad actors can game social proof metrics to induce the use of malign software. Here we study questions around the effects of such gaming using two field experiments on the largest developer platform, GitHub. To examine the impact of social proof, we bought 'stars' for a random set of GitHub repositories containing recently created Python packages, and estimated their impact on package downloads and broader repository activity. We find no discernible impact on downloads, nor on forks, pull requests, issues, or other measures of developer engagement. In our second field experiment, we manipulated the number of human downloads for Python packages. Again, we find no detectable effect on subsequent downloads or on any measure of repository activity. Our findings suggest that modest manipulation of social proof does not detectably shift developer adoption, though the threat may grow at higher manipulation intensities and in less-scrutinized contexts such as agentic coding, motivating platform signals that are harder to fake.

1 Introduction

Most commercial software relies on open-source software (Mojica et al. 2014; Eghbal 2016; Larios Vargas et al. 2020; Wermke et al. 2022), but vetting the quality of software with respect to security is hard (Munaiah et al. 2017). Hence, most software developers use cheap heuristics like social proof to choose between multiple open-source software that purport to solve the same problem (Vannoy and Palvia 2010; Pickerill et al. 2020). This raises the concern that bad actors can game social proof metrics to induce developers to use malicious software (Ohm et al. 2020; Cao and Dolan-Gavitt 2022; Ladisa et al. 2023; Wermke et al. 2022; Check Point Research 2024; Forbes 2024; He et al. 2024; Sonatype 2024). In this paper, we explore the possibility that faked social proof of software can induce popularity and usage using two field experiments.

doi:10.54501/jots.v3i2.286

Our first field experiment was conducted on GitHub, the world's largest developer platform. We manipulated the perceived popularity of a random set of repositories associated with new Python packages by purchasing 'stars' for them. For a random subset of the treated repositories, we further increased the 'stars' by asking people in our network to 'star' the repositories. In all, our manipulation raised the median number of stars from 0 to 20–69 stars, depending on the treatment group. This meaningful but modest increase in the number of stars, however, had little impact on the number of package downloads or contributions to the repository. We cannot reject the null hypothesis that the manipulation had no effect three months after the treatment.

In a second field experiment, we manipulated social proof of Python packages by increasing the download count in the official registry. We used a script to download a random set of packages multiple times, nearly quintupling the median download count from about 20 to 100. (In absolute terms, the treatment manipulation was still modest, and deliberately so.) But there was little impact of the treatment on the number of downloads or contributions to the associated GitHub repository four months later.[1]

Our study allays but does not dispel concerns about the possibility of spreading malware by gaming social proof. For one, even a single organization adopting a malign piece of software may constitute a practically important effect. However, our experiments are underpowered to detect anything so subtle, and it is nearly impossible to design experiments that can. For another, it is entirely plausible that a more intense treatment would persuade developers.

2 Social Proof

Others' choices, especially those of similar others or those we admire, can be informative (Rao, Greve, and Davis 2001; Cialdini 2001; Salganik, Dodds, and Watts 2006; Vannoy and Palvia 2010; Amblee and Bui 2011; Dabbish et al. 2012; Messing and Westwood 2014; Venema et al. 2020). They can tell us what is useful or desirable. For instance, if someone is in the market for a car, they may pay attention to the cars in the office parking lot to infer what a good car is for them. Companies understand the value of social proof and regularly use it to try to influence customers. For example, many companies prominently show lists of customers who have bought their products on their websites. Retailers like Amazon allow customers to sort by best sellers and show how many customers bought a product from a particular seller over the last month or year. News media companies show lists like 'most read articles' to aid readers (Messing and Westwood 2014).

1. In the Appendix, we report results from an analysis of whether bot and human downloads Granger-cause subsequent human downloads. The results suggest that human downloads predict future human downloads (Appendix C). However, because this pattern is inconsistent with the experimental findings, we are cautious about placing too much weight on it.

The Elaboration Likelihood Model (ELM) provides a framework for understanding when social proof is likely to influence decisions (Petty and Cacioppo 1984, 1986). The model distinguishes two routes of persuasion: a central route, in which decision makers carefully evaluate the merits of the object itself, and a peripheral route, in which they rely on heuristic cues such as popularity or endorsement counts. The peripheral route dominates when decision makers lack either the motivation or the ability to engage in careful evaluation. Social proof metrics—star counts, download tallies, best-seller rankings—function as peripheral cues, and their influence is predicted to be strongest when the decision maker cannot easily assess quality directly.

Signaling Theory offers a complementary perspective (Spence 1973). A signal is informative to the extent that it is costly or difficult to produce without possessing the underlying quality it purports to indicate. When a signal becomes cheap to produce regardless of quality, its correlation with quality degrades, and rational receivers who recognize this degradation discount the signal accordingly (Campbell 1979). Critically, receivers need not know the exact cost of producing a fake signal; it is sufficient that they observe or infer that the signal is noisy—for instance, by encountering cases where high signal values do not correspond to high quality.

Software adoption is an instructive case for both frameworks. Inspecting code for safety and quality takes time and skill, and most developers cannot afford to do so for every dependency they adopt. Social proof can therefore be especially influential when choosing between packages that purport to provide the same functionality (Dabbish et al. 2012; Vannoy and Palvia 2010; He et al. 2024). At the same time, the ELM predicts that developers may be less susceptible to peripheral cues than users of other platforms: developers face tangible consequences from poor choices (broken builds, security vulnerabilities, maintenance burden), giving them strong motivation to evaluate quality through the central route. They can also draw on richer signals than raw popularity, including code documentation, commit frequency, contributor activity, and project responsiveness (Dabbish et al. 2012; Tsay, Dabbish, and Herbsleb 2014; Borges and Tulio Valente 2018). From the perspective of Signaling Theory, the informational value of GitHub stars is compromised by commercial markets that sell stars at trivial cost. Vendors are easily located via standard web searches at prices as low as $0.10 per star (He et al. 2024). When the cost of faking a signal is low relative to its value, the signal's correlation with quality degrades, and receivers who encounter evidence of this, whether through direct awareness of these markets or through experiences with repositories that have a high star count yet are low-quality, have reason to discount it (Campbell 1979).

These predictions of attenuation are not unambiguous, however. Even when decision makers are aware that a cue is noisy, anchoring effects can cause high values to inflate quality judgments (Tversky and Kahneman 1974). A developer who is skeptical of stars may still perceive 500 stars differently from 5, because the number sets a reference point

that is difficult to fully adjust away from. Moreover, the ability to engage in central-route evaluation varies: assessing package safety and fitness for purpose is difficult without inspecting the code, and under pressure, developers may default to peripheral heuristics even when they would prefer not to. Whether social proof in the form of manipulated platform metrics actually shifts developer behavior is therefore an empirical question that the competing predictions above cannot resolve on their own.

To shed light on how manipulable developers' choices are, we conduct two field experiments. In the first, we manipulate the number of stars for a random set of new Python packages on GitHub and assess the impact on downloads of the associated package and activity on the repository. In the second, we manipulate the number of downloads of Python packages to assess its impact on future downloads and contributions to associated GitHub repositories.

3 GitHub Experiment

GitHub is the most popular platform for creating, storing, managing, and sharing code. Over 100 million developers use GitHub, and the platform hosts over 420 million repositories, of which over 280 million are public.[2]

GitHub prominently displays four social proof signals for each repository: the number of stars, watchers, forks, and open issues. These signals differ in what they convey and how readily they function as quality or popularity indicators. Forks indicate derivative work rather than endorsement, and open issues can signal either active use or neglect. Watching provides a clearer signal of interest, but relatively few users watch repositories because doing so subscribes them to notifications about repository activity. That leaves stars, an analog of 'likes,' as the most widely used metric of popularity and social proof on GitHub (Munaiah et al. 2017; Borges and Tulio Valente 2018; Qiu et al. 2019; Pickerill et al. 2020; Larios Vargas et al. 2020; Wermke et al. 2022; Koch, Klein, and Johns 2023; Shen 2023). Unlike other platforms, GitHub does not readily show visit counts or similar engagement metrics, making stars all the more salient.

Users star repositories for various reasons. Some use stars as a bookmarking tool; GitHub lets users browse, search, and organize their starred repositories into lists. Others star repositories to signal approval to their followers, since starred repositories appear in followers' news feeds. Still others star a repository simply to show support for a project or its maintainer. GitHub also uses stars to customize a user's news feed, recommending related repositories and surfacing changes to starred projects. Whatever the motivation, each star adds to a repository's visible social proof.

2. Figures as of January 2023; see GitHub Octoverse 2023 (https://web.archive.org/web/20240907070313/https://github.blog/news-insights/research/the-state-of-open-source-and-ai/).

Repository owners try hard to get people to star their repositories to increase visibility and hence usage. Starring can increase visibility through three channels. First, starred repositories appear in the news feeds of the starring user's followers, exposing the repository to a wider audience. Second, stars contribute to a repository's chances of appearing on GitHub's trending page,[3] which attracts additional attention, including from the media.[4] Third, a high star count serves as social proof: a developer evaluating the repository may be persuaded by the visible endorsement of others to adopt the software. The number of stars a repository has is widely considered the primary signal of its popularity, which is why it is the metric we intervene on. We believe our experiment operates primarily through the social proof channel rather than the news feed or trending channels, for two reasons. First, our manipulation is modest enough that it does not cause treated repositories to trend. Second, the accounts we use to add stars have follower networks whose interests are unrelated to the treated packages, limiting the informational value of the stars for those followers.

3.1 Sample and Randomization

Our population of interest is repositories associated with Python packages newly listed on the Python Package Index (PyPI) repository,[5] which hosts open-source Python packages uploaded by developers. We focus on new listings because we conjecture that quality is the least certain when a package first appears on the index, so social proof should provide the strongest signal. Note that a new listing does not imply a new codebase: the underlying repositories may have existed prior to listing. Our choice of Python packages stems from availability of reliable data on download counts through PyPI. Our sample includes packages listed between 24 and 30 April 2023, identified by taking the set difference of the PyPI index on those two dates. Of these packages, we keep only those with a public GitHub repository, linked via the GitHub source URL in each package's setup configuration file. In all, we identified 622 newly listed packages with a public GitHub repository. Of the 622, we assigned 100 to the treatment group.

3.2 Treatment Conditions

Our treatment includes stars from two sources: market-bought and network-based clicks. We asked users in our network to 'star' the 100 repositories. This group serves as our 'low dosage' treatment group. The median number of followers of users in our network who starred the treatment repositories was 9 (the mean was 64, Appendix A.1). We expect the benefits of these stars to also come primarily from greater social proof because the repositories that the users starred did not focus on the interests and specializations of the users. We triggered our network to 'star' the repositories on May 12, 2023. They took about 10 days to 'star' all the 100 repositories (see Figure 1).

3. https://github.com/trending.
4. While GitHub uses a combination of factors to determine trending repositories, stars are generally suspected as a key factor. See, for example, https://github.com/orgs/community/discussions/3083.
5. https://pypi.org/

Of the 100 packages, we randomly selected 25 packages and bought 50 stars for each from Baddhi Shop[6] on May 12, 2023. The stars we bought from the vendor were assigned over a few days, plausibly to avoid triggering GitHub's anomaly detection algorithms. One interesting feature of these purchased stars is that they were all from users whose accounts were created around April 20, 2023. We expect some of these stars to be taken down by GitHub integrity teams, so these stars likely only have a short-run impact. For the stars we bought, we have no reason to expect the users who starred the repository or their followers to be authentic. So, we only expect the benefits of these stars to accrue from people who look at the GitHub repository before downloading a package.

3.3 Attrition and Analytic Strategy

On May 20, 2023, about a week after the intervention started, we took a snapshot of the GitHub repositories. By that time, we could only retrieve 582 repositories (Section 3.4). Thirty-seven repositories in the control group and three from the treatment group were either deleted or moved to private status. It is plausible that the somewhat lower attrition in the treatment group is because the repository owners were encouraged by seeing more stars. Our primary estimand is Intent-to-Treat (ITT) effects on Python package downloads, for which we used the entire dataset of 622 Python packages.

3.4 Balance Tests

To validate our randomization, we compared pre-treatment outcomes and activity across the treatment and control groups. Looking at the primary outcome variable, downloads, in the pre-treatment period, we find little difference between treatment and control packages (see Figure 2). We also constructed pre-treatment balance tables using historical GitHub event data from the GitHub Archive Project (Appendix A.2). The pre-treatment activity metrics (stars, forks, pushes, pull requests, opened issues, closed issues, and releases) confirm balance between the treatment groups.

3.5 Manipulation Check

While historical download logs are immutable, users can rescind stars. Anticipating that bought stars might be flagged by GitHub and removed shortly after being added, we took a snapshot of the stars of each package at the end of every day. This allowed us to more accurately capture the change in stars during our experiment period.[7]

Figure 1 shows the time trend of the median number of stars across the three groups: high-dosage treatment (market and network stars; N = 25), low-dosage treatment (network stars only; N = 75), and control (no stars; N = 485). By the end of the intervention (May 21), relative to the control group, the median number of additional stars in the low-dosage

6. https://web.archive.org/web/20240619193418/https://baddhi.shop/product/buy-github-followers/.
7. Historical stars with timestamps are available from the API, but do not retain records of removed stars.

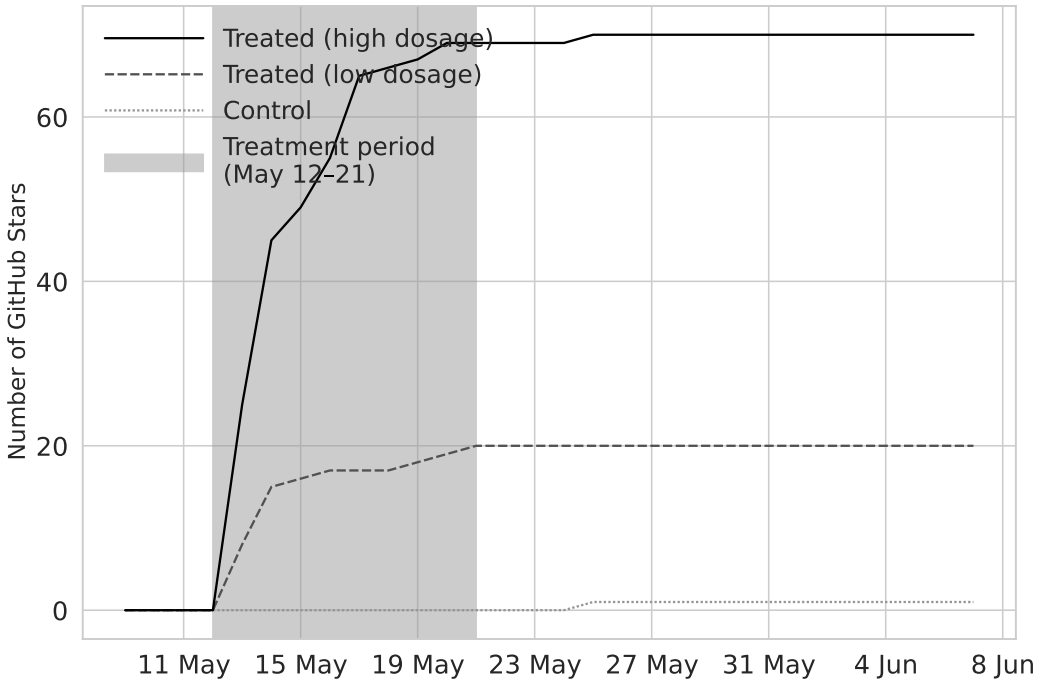

Figure 1: **Manipulation Check: GitHub Stars for Treated vs. Control**. The figure plots the median number of stars on a particular day for the three groups: high-dosage treatment (market and network stars), low-dosage treatment (network stars), and control (no stars). In all, we plot data for the initial 585 packages and 17,550 package days over the period May 9–June 7, 2023. The shaded vertical bar indicates the period during which the treatment was applied. In the Appendix, Table A.5 presents formal estimates of this manipulation check, and Figure A.1 shows the means.

group was 20 ($p < .001$), while the median number of additional stars in the high-dosage group was 69 ($p < .001$, Table A.5, Appendix).

3.6 Outcome Measure: PyPI Downloads

The primary outcome for both field experiments—the GitHub experiment and the PyPI experiment (Section 4)—is Python package downloads. These Python package download metrics come from the centralized PyPI repository. This resource hosts open-source Python packages uploaded to it by package developers, and users download packages from it as needed. For both experiments, we tracked daily downloads over the sample period April–October 2023.

To log package downloads from PyPI, the Linehaul project[8] implements a daemon that listens for download events and logs them. Specifically, it tracks details like the package name, date and time the package was downloaded, package version, and the type of installer software. Linehaul then feeds the download logs to the publicly available Google BigQuery. Some installers are known bots (Table B.2, Appendix). These bots tend to be caching mirrors for distributional and security purposes. For instance, Bandersnatch is the official mirroring service of PyPI to help improve accessibility and download speed

8. https://github.com/pypi/linehaul-cloud-function.

for users of various geographical regions. We restricted the analysis to human downloads (Table B.2, Appendix).

The main analyses focus on differences in medians, given the huge volatility in means induced by a few extreme outliers (Figures A.2 to A.4, Appendix).

3.7 Results

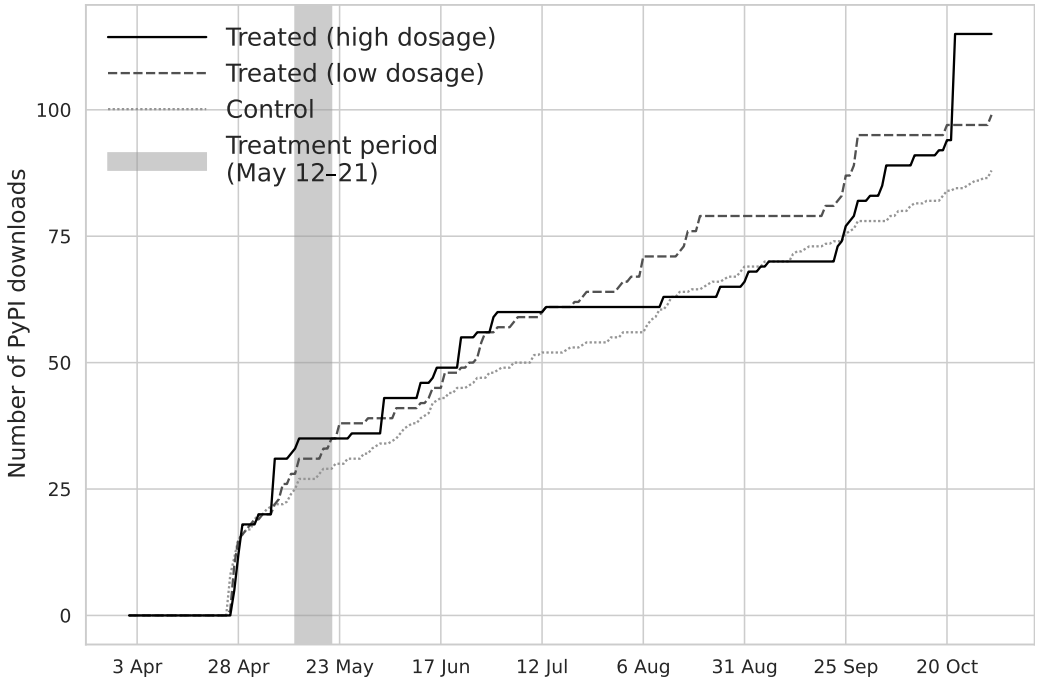

Figure 2: **Median PyPI Downloads for Treatment vs. Control in GitHub Experiment.** The figure plots the median of cumulative downloads for each of the three groups. Each point is a day averaged within the group for 622 packages and 133,108 package days over the sample period April–October 2023. Treatment is distinguished by low and high dosage (see Section 3.2). The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table B.2, Appendix). In the Appendix, Figures A.3 to A.4 show the time series of individual packages, Table A.6 reports estimates of differences in medians, and Figure A.2 plots the means.

Figure 2 traces the median number of PyPI downloads over time. If the treatment shifted adoption behavior, we would expect the download trajectories of treated and control packages to diverge after intervention. No such divergence is apparent. The ITT estimates in Table A.6, Appendix confirm this. One month after intervention (on June 20), the median number of downloads in the low-dosage and high-dosage groups was statistically indistinguishable from the control group. We estimated differences in medians at four additional time points through October 2023 and found no significant difference between any treatment group and the control at any snapshot. We also estimated a model allowing treatment effects to vary linearly over time. Neither the low-dosage nor the high-dosage group exhibits a trend distinguishable from that of the control group (see Table A.6,

Appendix). In Appendix A.4.2, we examine differences in means and reach similar conclusions.[9,10]

Additionally, to test whether packages with greater technical complexity benefit more from social proof, we tested if treatment effects varied by package complexity, proxied by repository size (total size of all files in MB) and README documentation length (Appendix A.4.3). We found no significant interactions at any post-treatment snapshot date; however, the confidence intervals on the interaction terms are sufficiently wide that we cannot rule out meaningfully large differential effects. We return to the power limitations of these tests in the discussion.

Finally, we tested whether the increase in stars spilled over into broader GitHub activity on treated repositories. Using post-treatment data from GitHub Archive, we estimated treatment effects on six additional metrics: forks, pushes, pull requests, issues opened, issues closed, and releases. We found no significant differences between treated and control groups on any metric (Appendix A.5).

4 PyPI Experiment

We experimentally manipulated human downloads and tested the hypothesis that higher human downloads lead to yet higher future human downloads.[11]

4.1 Design

We randomly sampled 50,000 packages from the PyPI repository and filtered to those with at least five human downloads (Section 3.6). This left us with 23,965 packages. We then randomly assigned 20% of the packages to the treatment group (N = 4,823) and the remaining 80% to the control group (N = 19,142). As with the GitHub experiment, we tracked daily downloads over the same sample period (April–October 2023) and confirmed pre-treatment balance across various GitHub activity metrics (Appendix B.1). For packages in the treatment group, we wrote a script to download the same package 100 times. We downloaded the package in a way that each download shows up in the official Linehaul numbers (Section 3.6).[12] We treated the packages between June 3, 2023, and June 8, 2023. Figure 3 serves as our manipulation check, with the gray region indicating the sharp increase in downloads within the treatment period.

9. Subsetting downloads to those installed by *pip* (Table B.2, Appendix), the most common human installer, yields similar results.

10. We also tested whether the manufactured stars in the treatment groups attracted additional organic stars beyond those we added and found no evidence of this (untabulated).

11. We also analyzed if human downloads Granger-cause human downloads (Appendix C), and found they do. However, we are cautious about overinterpreting these results, as they conflict with the experimental findings.

12. When installing packages, most user systems usually cache the source download files (e.g., `.tar.gz`, etc.) on the local drive to save on bandwidth resources. Cognizant of this, we wrote the script so that it installs packages without using the cached download directory. This option forces the process to always download the package from PyPI instead of using the cached source files.

4.2 Results

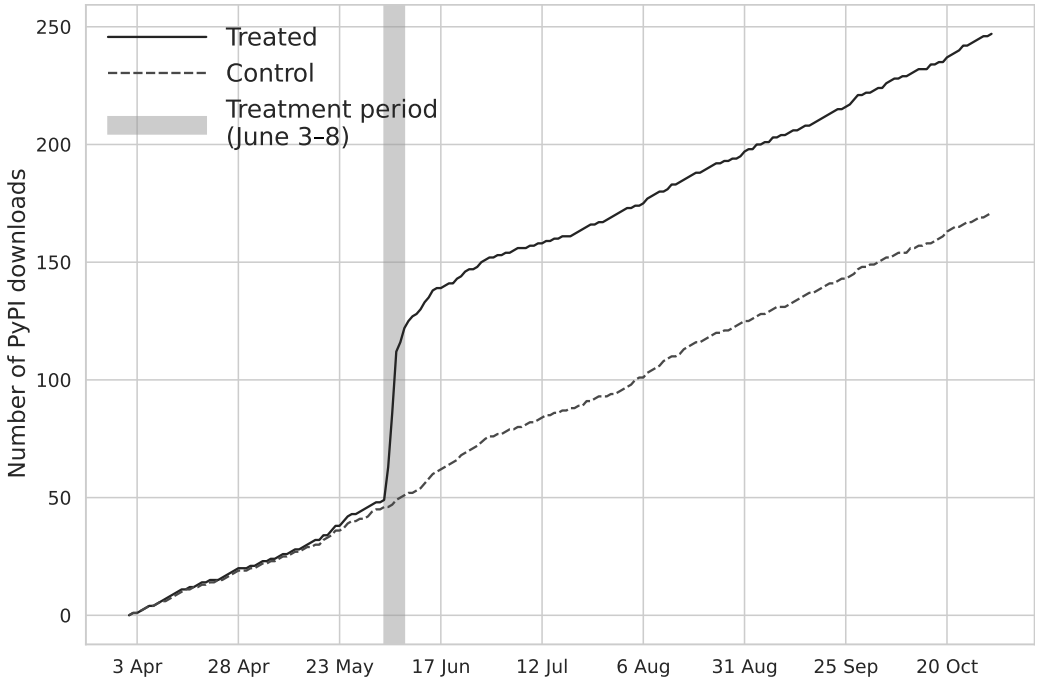

Figure 3: **Median PyPI downloads for Treated vs. Control in the PyPI Experiment.** The figure shows trends in median daily downloads for the treated packages (N = 4,823) and control group packages (N = 19,142) for 5,128,510 package-day observations over the sample period April–October 2023. The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table B.2, Appendix). In the Appendix, Figure B.1 shows the same figure for mean downloads, and Table B.3 reports the estimates for the differences in medians.

Figure 3 plots median cumulative downloads for the treatment and control groups. Pre-treatment, little separated the two series. During the treatment period (June 3–8), the series diverged, and by July 8—a month later—the median treated package had 75 more cumulative downloads than the median control package ($p < .001$, column (1) of Table B.3, Appendix). This difference was approximately equal to the 100 downloads our script mechanically added, confirming the manipulation.

The behavioral question is whether the inflated count attracted additional organic downloads—that is, whether the treated-control gap widens beyond the mechanical increment. Columns (1)–(4) of Table B.3, Appendix show the treated-control gap at monthly snapshots across four months post-treatment: the gap is stable at approximately 72–75 downloads throughout, with no sign of widening or narrowing. Column (5) estimates a trend model over the full 145-day post-treatment period (Jun 8–Oct 31, 2023). The treatment-by-trend interaction is -0.0 downloads per day (p = .657; 95% CI: [−0.1, 0.1]), consistent with a flat gap and no behavioral amplification at the median. The mean estimates tell a consistent story, though with far less precision: the trend interaction is 849 downloads per day (p = .621; 95% CI: [−2,513, 4,211], column (5) of Table B.4, Appendix).

Finally, we tested whether the increase in downloads spilled over into GitHub activity. Manipulated download counts on PyPI did not produce higher numbers of stars, forks, pushes, pull requests, issues, or releases in the corresponding GitHub repositories (Appendix B.4).

4.3 Precision of Estimates

To interpret the null results, we characterize what effect sizes our estimates can and cannot exclude, using confidence intervals on the behavioral estimand rather than post hoc power calculations.

For the median outcome—the paper's primary estimand—the result is unambiguous. The treatment-by-trend interaction in column (5) of Table B.3, Appendix has a 95% confidence interval of $[-0.1, 0.1]$ downloads per day, spanning zero. Over the 145-day post-treatment window, the monthly snapshot estimates (columns (1)–(4)) are stable at 72–75 downloads, bracketing the mechanical increment. At the median, we can rule out any sustained positive behavioral amplification.

For the mean outcome, the confidence intervals are far wider, driven by extreme right-skew in the download distribution. The treatment-by-trend interaction in column (5) of Table B.4, Appendix has a 95% confidence interval of $[-2{,}513, 4{,}211]$ downloads per day. The upper bound implies that we cannot exclude cumulative mean behavioral effects as large as approximately $4{,}211 \times 145 \approx 611{,}000$ downloads over the post-treatment window. Equivalently, the 95% confidence interval on the treatment level shift at one month (column (1)) extends to 417,223 downloads, approximately 163% of the mean outcome at that snapshot. These wide intervals reflect the heavy-tailed nature of the download distribution rather than a deficiency of sample size: our experiment includes nearly 24,000 packages and over five million package-day observations, but a handful of popular packages dominate the mean.

In sum, the median estimates provide tight evidence against behavioral amplification. The mean estimates cannot exclude large effects, but this imprecision is inherent in the outcome distribution rather than remediable through larger samples at this treatment intensity. Our design prioritized ethical constraints and ecological realism, which limits our ability to detect effects that are small in the aggregate but concentrated among a few high-value adoption events, or effects that would emerge under more intense or sustained manipulation.

5 Discussion

Many existing studies examine how people use cheap decision-making heuristics in technology adoption (Vannoy and Palvia 2010; Pickerill et al. 2020). Separately, a growing body of research has documented the rise in malware delivered through dependencies

within the open-source software supply chain (Ohm et al. 2020; Cao and Dolan-Gavitt 2022; Ladisa et al. 2023; Wermke et al. 2022; Check Point Research 2024; Forbes 2024; He et al. 2024; Sonatype 2024). He et al. (2024) find that most faked stars on GitHub are used to promote disguised malware. We contribute to this literature by conducting two field experiments to evaluate whether artificially inflated social proof influences software adoption. In both experiments, we find no detectable effect. Manipulating star counts on GitHub did not increase downstream package downloads, and manipulating download counts on PyPI did not increase subsequent organic downloads. These null results obtain at the treatment intensities our ethical constraints permitted; we cannot rule out effects at the scale that commercial astroturfing campaigns deploy.

Our findings contrast with those of prior studies (Salganik, Dodds, and Watts 2006; Fang et al. 2022), including field experiments (Muchnik, Aral, and Taylor 2013; Rijt et al. 2014). In an artificial music market, Salganik, Dodds, and Watts (2006) find that displaying download counts amplifies the success of higher-ranked songs. Rijt et al. (2014) conducted field experiments across four platforms (Kickstarter, Epinions, Wikipedia, and Change.org), finding that small early endorsements increase the likelihood of subsequent success. Muchnik, Aral, and Taylor (2013) find that manipulated upvotes on a social news aggregation site increased subsequent positive ratings, demonstrating herding effects, although effects were topic-dependent and critically absent for IT-related posts. In a non-randomized natural experiment on GitHub, Fang et al. (2022) find that tweets promoting repositories attract more stars, though the effect is weaker when the tweet author is affiliated with the promoted repository. A key difference between our setting and these prior studies is that software adoption involves a consequential technical decision—integrating a dependency into a codebase—where the cost of a poor choice (broken builds, security vulnerabilities, maintenance burden) provides strong motivation to evaluate quality directly rather than rely on peripheral cues.

Software adoption may differ from these social settings in ways that attenuate the influence of peripheral metrics. The Elaboration Likelihood Model (Petty and Cacioppo 1984, 1986) predicts that peripheral cues—star and download counts—carry the most weight when decision makers lack either the motivation or ability to evaluate the underlying quality. Developers choosing a new package for a project are motivated to evaluate quality and have at least some ability to do so through code documentation, commit frequency, contributor activity, and project responsiveness (Dabbish et al. 2012; Tsay, Dabbish, and Herbsleb 2014; Borges and Tulio Valente 2018), and thus may give less weight to peripheral metrics. The Muchnik, Aral, and Taylor (2013) IT-null result is consistent with this account: IT-literate users, like developers, may process content via the central route rather than deferring to social cues. The finding by Fang et al. (2022) that signal source credibility moderates developer responses to endorsements further supports the view that developers are not passively influenced by peripheral metrics.

Signaling Theory (Spence 1973) offers a complementary lens. Signals lose credibility when they are easy to fake. GitHub stars are commercially available at scale for trivial cost, and these markets are readily accessible. As He et al. show, these vendors are easily found via standard web searches, with prices as low as $0.10 per star, giving informed developers reason to discount star counts as reliable quality indicators (Campbell 1979; He et al. 2024).

The ability to evaluate quality varies considerably across contexts, however. Assessing package safety, maintenance trajectory, and fitness for purpose remains difficult without downloading and inspecting the code, and sometimes even after doing so. In agentic coding workflows, where LLM-based tools select and install packages with limited human oversight, the scrutiny that might attenuate social proof effects is largely absent. Conversely, in corporate environments, organizations increasingly rely on software composition analysis tools and internal registries that gate which packages developers can install, substituting institutional review for individual judgment. The role of peripheral signals like stars is therefore neither uniformly decisive nor uniformly irrelevant, but depends on the decision-making environment. Our null results should not be read as evidence that social proof is irrelevant to software adoption, but rather that a single manipulated metric, at modest intensity, did not detectably shift behavior in our sample.

5.1 Implications for Platform Design

These findings nonetheless carry practical implications. Even if stars did not shift adoption in our experiments, the threat of social proof manipulation persists as manipulation scales up and as decision-making contexts shift toward less scrutinized environments. Several concrete platform interventions could reduce the efficacy of such manipulation. First, GitHub could surface richer contextual information alongside star counts: star velocity over time (since sudden spikes are a common manipulation signature), the proportion of stargazers with substantive commit histories, and the account age distribution of stargazers. Second, platforms could more prominently integrate composite security health metrics. The OpenSSF Scorecard, for instance, aggregates automated checks across code review practices, dependency management, vulnerability disclosure, and CI/CD configuration into a single risk-weighted score, offering a harder-to-fake alternative to raw popularity counts. Third, the emerging infrastructure of build provenance and trusted publishing—now supported by PyPI, npm, RubyGems, and NuGet via OpenID Connect attestations—provides cryptographic verification of where and how a package was built, shifting trust from social signals to verifiable supply chain evidence. Fourth, beyond removing fake stars after the fact, GitHub could impose graduated consequences on repositories that engage in coordinated manipulation, including freezing or flagging repositories, particularly those distributing malicious payloads (He et al. 2024). Such enforcement reduces the expected payoff to astroturfing regardless of whether individual developers attend to these signals.

5.2 Ethical Considerations

Like other field experiments (Muchnik, Aral, and Taylor 2013; Rijt et al. 2014), this study implements manipulation of publicly visible metrics on live, real-world platforms without prior consent from users—a design choice necessary for ecological validity. Our protocol is consistent with the *ACM Publications Policy on Research Involving Human Participants and Subjects*, the *ACM Code of Ethics and Professional Conduct*, and the principles from the *Menlo Report* (Association for Computing Machinery 2021, 2018; Bailey et al. 2012). As Kohno, Acar, and Loh (2023) argue, IRB approval alone does not constitute sufficient ethical justification; we therefore draw on the consequentialist framework to assess whether the anticipated benefits of our design outweigh its potential harms.

The potential harms were narrow in scope and limited in duration. We added stars to newly published, low-visibility repositories and incremented download counts for a bounded period (see Figure 3). We did not collect personal or sensitive data, alter code, documentation, or dependencies, deploy malicious content, or expose users to security risks. The number of stars added was large enough to be visible for packages with minimal prior exposure, but modest enough to avoid triggering trending algorithms or drawing unusual scrutiny. Similarly, in the PyPI experiment, scripted downloads were sufficient to shift download tallies but unlikely to meaningfully mislead users or affect rankings at scale. Any reputational effect on treated repositories was small and temporary.[13]

The potential benefits, by contrast, are substantial. Supply chain attacks through manipulated social proof represent a documented and growing threat (Ohm et al. 2020; Ladisa et al. 2023; Ohm and Stuke 2023; He et al. 2024; Sonatype 2024). Establishing whether such manipulation actually shifts developer behavior is a prerequisite for designing effective countermeasures—and this question cannot be answered without in situ experimentation on real platforms.

We nonetheless acknowledge the deontological tension inherent in this design: package maintainers were not consented as experimental units, a concern not dissolved by the null result. The conservative design choices described above represent our effort to minimize rights violations within the constraint that full prior consent would destroy ecological validity.

Relatedly, we did not debrief package maintainers after the study, as this risked causing reputational concern or defensive actions (e.g., deleting the repository) disproportionate to the treatment itself, a tradeoff that comparable field experiments on live platforms have also faced (Muchnik, Aral, and Taylor 2013; Rijt et al. 2014).[14]

13. In a follow-up, we confirm that the median star count for the high-dosage repositories had fallen from ~70 in the immediate post-treatment window (see Figure 1) to 22, consistent with the platform's eventual removal of the 50 purchased stars after we bought them in May 2023.

14. A related concern is that maintainers of low-traffic treated packages might have interpreted the added stars as a genuine signal of demand and invested additional unwarranted effort in their package. We find no empirical evidence of this, as we observe no discernible differences in push events (and other activities such as pull requests, and the opening and closing of issues) between treated and control groups (Appendix A.5).

5.3 Limitations

The ethical constraints of a limited treatment scope, while necessary, have implications for statistical power. Although our analysis (Section 4.3) provides evidence against large effects, we cannot exclude the possibility that a more intense or sustained manipulation—on the scale that commercial astroturfing campaigns would deploy—would produce a measurable behavioral change in developers. Real astroturfing campaigns maintain and escalate signals over time, whereas our treatment was a one-shot boost whose salience may decay in ways that sustained manipulation would not. Moreover, while we test for treatment effects that vary linearly over time, which likewise yield nulls, identifying dose-response non-linearities would require multiple treatment-intensity arms, and detecting threshold effects in time would require greater temporal granularity and statistical power than our experiment affords (Gelman, Hill, and Vehtari 2020).

We also tested for heterogeneous treatment effects by package complexity using repository size and documentation length as proxies, under the hypothesis that peripheral cues like stars carry more weight when direct evaluation is harder (Petty and Cacioppo 1984, 1986). We found no significant interactions; however, the confidence intervals on the interaction terms are wide enough that we cannot reject substantively meaningful differential effects (see Tables A.9 to A.10, Appendix). Our sample size provides limited statistical power to detect interactions, and would have to be roughly 16 times larger for main effects under reasonable assumptions about relative effect sizes (Gelman, Hill, and Vehtari 2020). The absence of significant heterogeneity is therefore consistent with both a genuinely uniform null and with meaningful moderation that our design is underpowered to detect. These proxies are also crude measures of how difficult packages are to evaluate. Metrics that more directly capture evaluability, such as cyclomatic complexity, dependency depth, or test coverage, would provide sharper tests of the ELM prediction that peripheral cues matter most when direct assessment is difficult.

A further threat to validity concerns the outcome measure itself. We filter PyPI downloads to remove those by bot installers (Section 3.6), but it remains possible that our measures include downloads from automated CI/CD pipelines and container builds that install packages as part of testing and deployment workflows. This concern is mitigated by the fact that our sample consists of newly published, low-visibility packages that are unlikely to feature in many automated build pipelines. In any case, such automated downloads would not respond to star manipulation, so any residual contamination would attenuate treatment effects and bias toward the null rather than generate spurious effects.

Finally, the scope of our experiments limits generalizability. Both studies focus on newly published and low-visibility Python packages. Peripheral heuristics like star counts are theorized to be most influential precisely when richer evaluation signals are absent (Petty and Cacioppo 1984, 1986)—which is the condition our sample reflects. Our null results are therefore unlikely to be reversed for established packages with longer track records and more credible social proof, though stars may function differently in that context,

serving as coordination signals or search-ranking inputs rather than as quality cues. The findings are also specific to the Python ecosystem and the 2023 period. Behavioral effects may differ across other language ecosystems and over time as platforms evolve.

5.4 Future Directions

Several directions for future research follow. First, larger doses within ethical constraints could help address whether the null effects would hold under more intense manipulation. We note, however, that a recurring tension in this line of research is that ecologically valid field experiments necessitate live-platform manipulation without participant consent. One path toward better deontological alignment (Kohno, Acar, and Loh 2023) might be opt-in experimental registries, where package maintainers and platform users consent in advance to participation. Another possible design is pairs of the same simulated but safe packages with varied manipulated metrics (and disguised names) to track differences in uptake without the ethical costs of live-platform intervention. Conjoint or discrete choice experiments offer yet another avenue: developers could be presented with hypothetical package profiles varying in stars, downloads, documentation quality, and maintenance recency, allowing researchers to estimate the marginal weight placed on each attribute without any platform manipulation at all. Such survey-experimental designs sacrifice ecological validity but permit far larger samples, sharper identification of interaction effects, and full control over the signal environment.

Second, the rise of agentic coding workflows introduces a qualitatively different decision-making process that merits dedicated investigation. When LLM-based tools select and install packages, the adoption decision is mediated by a model's training data and retrieval context rather than by a developer's direct evaluation of code quality. Studies that present coding agents with package selection tasks under experimentally varied star counts and download metrics could isolate whether these tools are more susceptible to peripheral cues than human developers. Such studies avoid the ethical costs of live manipulation entirely, since the decision maker is a model rather than an uninformed human participant.

Finally, future studies should examine other language ecosystems (e.g., npm), where supply chain attacks have been increasingly documented (Ohm et al. 2020; Ladisa et al. 2023; Ohm and Stuke 2023; Sonatype 2024). Cross-ecosystem comparisons are particularly valuable because platforms differ in how prominently they surface social proof metrics and in the maturity of their institutional safeguards, such as software composition analysis tools and curated registries, offering natural variation in the conditions under which peripheral cues may influence adoption.

5.5 Conclusion

We close with a note of caution, mindful of Carl Sagan's famous aphorism: "The absence of evidence is not evidence of absence." The threat persists as long as social proof

metrics remain gameable and corruptible, and people are insufficiently aware of their manipulability. As a recent study (He et al. 2024) shows, the scale of fake stars on GitHub is in the millions across thousands of repositories between 2019 and 2024. Malware on PyPI is rising as Python packages are increasingly exploited through dependency-based attacks (Ohm and Stuke 2023). The growing adoption of agentic coding workflows, in which LLM-based tools select and install packages with limited human oversight, may further reduce the scrutiny that individual developers apply to adoption decisions, potentially increasing the efficacy of astroturfing at precisely the moment when its consequences are most severe. These trends highlight the threat that astroturfing poses to online security.

References

Amblee, Naveen, and Tung Bui. 2011. "Harnessing the Influence of Social Proof in Online Shopping: The Effect of Electronic Word of Mouth on Sales of Digital Microproducts." *International Journal of Electronic Commerce* 16 (2): 91–114.

Association for Computing Machinery. 2018. "ACM Code of Ethics and Professional Conduct." Accessed August 20, 2025. https://www.acm.org/code-of-ethics.

———. 2021. "ACM Publications Policy on Research Involving Human Participants and Subjects." Accessed August 20, 2025. https://www.acm.org/publications/policies/research-involving-human-participants-and-subjects.

Bailey, Michael, David Dittrich, Erin Kenneally, and Douglas Maughan. 2012. "The Menlo Report." *IEEE Security & Privacy* 10 (2): 71–75. https://doi.org/10.1109/MSP.2012.52.

Borges, Hudson, and Marco Tulio Valente. 2018. "What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform." *Journal of Systems and Software* 146:112–29. https://doi.org/10.1016/j.jss.2018.09.016.

Campbell, Donald T. 1979. "Assessing the Impact of Planned Social Change." *Evaluation and Program Planning* 2 (1): 67–90. https://doi.org/10.1016/0149-7189(79)90048-X.

Cao, Alan, and Brendan Dolan-Gavitt. 2022. "What the Fork? Finding and Analyzing Malware in GitHub Forks." In *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb),* 1–8. https://doi.org/10.14722/madweb.2022.23001.

Check Point Research. 2024. "Stargazer Ghost Network." Accessed January 9, 2024. https://research.checkpoint.com/2024/stargazers-ghost-network/.

Cialdini, Robert B. 2001. *Influence: Science and Practice.* 4th. Boston, MA: Allyn & Bacon.

Dabbish, Laura, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository." In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work,* 1277–86. https://doi.org/10.1145/2145204.2145396.

Eghbal, Nadia. 2016. *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure.* New York, NY, USA: Ford Foundation.

Fang, Hongbo, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. 2022. ""This Is Damn Slick!": Estimating the Impact of Tweets on Open Source Project Popularity and New Contributors." In *Proceedings of the 44th International Conference on Software Engineering,* 2116–29. https://doi.org/10.1145/3510003.3510121.

Forbes. 2024. “Rising Threat: Understanding Software Supply Chain Cyberattacks and Protecting against Them.” Accessed January 9, 2024. https://www.forbes.com/councils/forbestechcouncil/2024/02/06/rising-threat-understanding-software-supply-chain-cyberattacks-and-protecting-against-them/.

Gelman, Andrew, Jennifer Hill, and Aki Vehtari. 2020. *Regression and Other Stories.* Cambridge: Cambridge University Press. https://doi.org/10.1017/9781139161879.

He, Hao, Haoqin Yang, Philipp Burckhardt, Alexandros Kapravelos, Bogdan Vasilescu, and Christian Kästner. 2024. “4.5 Million (Suspected) Fake Stars in GitHub: A Growing Spiral of Popularity Contests, Scams, and Malware.” Version 1, *arXiv,* https://doi.org/10.48550/arXiv.2412.13459.

Koch, S., D. Klein, and M. Johns. 2023. “The Fault in Our Stars: An Analysis of GitHub Stars as an Importance Metric for Web Source Code.” In *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb),* 1–8.

Kohno, Tadayoshi, Yasemin Acar, and Wulf Loh. 2023. “Ethical Frameworks and Computer Security Trolley Problems: Foundations for Conversations.” In *32nd USENIX Security Symposium (USENIX Security 23),* 5145–62. https://www.usenix.org/conference/usenixsecurity23/presentation/kohno.

Ladisa, Piergiorgio, Henrik Plate, Matias Martinez, and Olivier Barais. 2023. “SoK: Taxonomy of Attacks on Open-Source Software Supply Chains.” In *2023 IEEE Symposium on Security and Privacy (SP),* 1509–26. https://doi.org/10.1109/SP46215.2023.10179304.

Larios Vargas, Enrique, Maurício Aniche, Christoph Treude, Magiel Bruntink, and Georgios Gousios. 2020. “Selecting Third-Party Libraries: The Practitioners’ Perspective.” In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,* 245–56. https://doi.org/10.1145/3368089.3409711.

Messing, Solomon, and Sean J. Westwood. 2014. “Selective Exposure in the Age of Social Media: Endorsements Trump Partisan Source Affiliation When Selecting News Online.” *Communication Research* 41 (8): 1042–63. https://doi.org/10.1177/0093650212466406.

Mojica, Israel J., Bram Adams, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, and Ahmed E. Hassan. 2014. “A Large-Scale Empirical Study on Software Reuse in Mobile Apps.” *IEEE Software* 31 (2): 78–86. https://doi.org/10.1109/MS.2013.142.

Muchnik, Lev, Sinan Aral, and Sean J. Taylor. 2013. “Social Influence Bias: A Randomized Experiment.” *Science* 341 (6146): 647–51. https://doi.org/10.1126/science.1240466.

Munaiah, Nuthan, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. "Curating GitHub for Engineered Software Projects." *Empirical Software Engineering* 22 (6): 3219–53. https://doi.org/10.1007/s10664-017-9512-6.

Ohm, Marc, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks." In *Detection of Intrusions and Malware, and Vulnerability Assessment,* edited by Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves, 23–43.

Ohm, Marc, and Charlene Stuke. 2023. "SoK: Practical Detection of Software Supply Chain Attacks." In *Proceedings of the 18th International Conference on Availability, Reliability and Security.* https://doi.org/10.1145/3600160.3600162.

Petty, Richard E., and John T. Cacioppo. 1984. "The Effects of Involvement on Responses to Argument Quantity and Quality: Central and Peripheral Routes to Persuasion." *Journal of Personality and Social Psychology* 46 (1): 69–81. https://doi.org/10.1037/0022-3514.46.1.69.

———. 1986. *Communication and Persuasion: Central and Peripheral Routes to Attitude Change.* New York: Springer-Verlag.

Pickerill, Peter, Heiko Joshua Jungen, Mirosław Ochodek, Michał Maćkowiak, and Miroslaw Staron. 2020. "PHANTOM: Curating GitHub for Engineered Software Projects Using Time-Series Clustering." *Empirical Software Engineering* 25 (4): 2897–929. https://doi.org/10.1007/s10664-020-09825-8.

Qiu, Huilian Sophie, Yucen Lily Li, Susmita Padala, Anita Sarma, and Bogdan Vasilescu. 2019. "The Signals That Potential Contributors Look For When Choosing Open-Source Projects." *Proc. ACM Hum.-Comput. Interact.* 3 (CSCW). https://doi.org/10.1145/3359224.

Rao, Hayagreeva, Henrich R. Greve, and Gerald F. Davis. 2001. "Fool's Gold: Social Proof in the Initiation and Abandonment of Coverage by Wall Street Analysts." *Administrative Science Quarterly* 46 (3): 502–26.

Rijt, Arnout van de, Soong Moon Kang, Michael Restivo, and Akshay Patil. 2014. "Field Experiments of Success-Breeds-Success Dynamics." *Proceedings of the National Academy of Sciences* 111 (19): 6934–39. https://doi.org/10.1073/pnas.1316836111.

Salganik, Matthew J., Peter Sheridan Dodds, and Duncan J. Watts. 2006. "Experimental Study of Inequality and Unpredictability in an Artificial Cultural Market." *Science* 311 (5762): 854–56.

Seabold, Skipper, and Josef Perktold. 2010. *statsmodels: Econometric and Statistical Modeling with Python.*

Shen, Lucas. 2023. "Does Working from Home Work? A Natural Experiment from Lockdowns." *European Economic Review* 151. https://doi.org/10.1016/j.euroecorev.2022.104323.

Sonatype. 2024. "Open Source Supply, Demand, and Security." Accessed January 9, 2024. https://www.sonatype.com/state-of-the-software-supply-chain/open-source-supply-and-demand.

Spence, Michael. 1973. "Job Market Signaling." *The Quarterly Journal of Economics* 87 (3): 355–74.

Tsay, Jason, Laura Dabbish, and James Herbsleb. 2014. "Influence of Social and Technical Factors for Evaluating Contribution in GitHub." In *Proceedings of the 36th International Conference on Software Engineering,* 356–66. https://doi.org/10.1145/2568225.2568315.

Tversky, Amos, and Daniel Kahneman. 1974. "Judgment under Uncertainty: Heuristics and Biases." *Science* 185 (4157): 1124–31. https://doi.org/10.1126/science.185.4157.1124.

Vannoy, Sandra A., and Prashant Palvia. 2010. "The Social Influence Model of Technology Adoption." *Commun. ACM* 53 (6): 149–53. https://doi.org/10.1145/1743546.1743585.

Venema, Tina A. G., Floor M. Kroese, Jeroen S. Benjamins, and Denise T. D. De Ridder. 2020. "When in Doubt, Follow the Crowd? Responsiveness to Social Proof Nudges in the Absence of Clear Preferences." *Frontiers in Psychology* 11.

Wermke, Dominik, Noah Wöhler, Jan H. Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. 2022. "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects." In *2022 IEEE Symposium on Security and Privacy (SP),* 1880–96. https://doi.org/10.1109/SP46214.2022.9833686.

Authors

Lucas Shen, Institute for Human Development and Potential, Agency for Science, Technology and Research (lucas@lucasshen.com).

Gaurav Sood, Independent Researcher (gsood07@gmail.com).

Daniel Weitzel, Colorado State University (weitzel.daniel@gmail.com).

Acknowledgments

The authors thank the editors and reviewers for their valuable feedback.

Data availability statement

Code and data are published under an open-source license at https://github.com/themains/social_proof_stars, and archived at https://doi.org/10.5281/zenodo.19382900.

Funding statement

Not applicable.

Ethical standards

This study was reviewed by the Colorado State University Institutional Review Board (CSU IRB Protocol 7411). The authors have no conflicts of interest to declare.

Keywords

Field Experiment; RCT; GitHub; Security; Malware; Open-Source Software; Social Proof; Social Computing.

Appendices

Table of Contents

Appendix A: GitHub Experiment

A.1: Organic GitHub Starrers

This section lists the 20 users from the authors' network for the GitHub experiment. Table A.1 provides descriptive statistics on these users, who are listed in Table A.2.

Table A.1: **Summary statistics of organic starrers' GitHub characteristics.** This table shows summary statistics of the organic GitHub star givers listed in Table A.2. The last five rows are indicator variables.

	(1) Mean (SD)	(2) Median [IQR]
Public repositories	35.0 (49.2)	12.0 [6.0,61.0]
Public gists	9.4 (29.4)	0.0 [0.0,4.0]
Followers	63.6 (160.4)	9.0 [1.0,17.0]
Following	34.7 (90.5)	7.0 [0.0,20.0]
Account age (years)	8.1 (3.5)	8.5 [6.1,10.4]
Lists name	0.9 (0.4)	—
Lists company	0.4 (0.5)	—
Lists blog/website	0.6 (0.5)	—
Lists geographical location	0.5 (0.5)	—
Lists brief biography	0.5 (0.5)	—
Lists Twitter handle	0.3 (0.5)	—

Table A.2: **List of GitHub organic starrers.** Table lists the 20 organic GitHub star givers (see Section 3.2). See Table A.1 for summary statistics of these starrers.

(1) Username	(2) Created	(3) Repos	(4) Gists	(5) Followers	(6) Following
basharnaji	2014-05-14	9	0	9	10
bmwhetter	2016-09-29	3	0	1	1
BonaventureR	2017-09-20	6	0	3	3
c-s-ale	2016-06-02	28	0	13	2
chris-alexiuk	2022-09-26	19	0	10	0
danweitzel*	2013-01-15	11	5	13	25
deepankermishra	2014-10-17	5	0	5	9
dhingratul	2013-06-21	82	0	95	64
dhruvarora-db	2021-07-20	0	0	0	0
dwillis	2008-03-04	208	134	705	130
humanpranav	2020-07-15	8	0	0	0
khurramnasser	2012-07-09	0	0	0	0
LSYS*	2015-01-13	13	4	17	407
Madhu009	2016-06-07	61	0	186	7
NoahFinberg	2014-08-27	12	4	10	20
pvbhanuteja	2017-10-30	61	12	1	0
rajashekar	2011-10-20	67	9	23	33
sharmaamits	2017-12-19	1	0	0	0
sjhangiani12	2015-05-25	41	0	5	8
soodoku*	2011-04-11	93	29	239	10

Note: Asterisks (*) denote the authors' own GitHub accounts.

A.2: Balance Tests Using GitHub Archive Events

In this section, we constructed pre-treatment balance tables using historical GitHub event data from the GitHub Archive Project. We aggregated cumulative event counts for each repository from January 1, 2023, to the start of each experiment's treatment period (May 12 for the GitHub experiment) across seven GitHub activity metrics: stars (WatchEvent), forks, pushes, pull requests, issues opened, issues closed, and releases.

For the GitHub experiment, we observe no pre-treatment difference between the treated and control groups (Table A.3). When we split the treatment group by dosage (high vs. low), we observe a pre-treatment difference between the high-dosage group and the control group on issues and forks (Table A.4). Including these seven pre-treatment metrics as baseline covariates does not change the null findings (see Table A.6). However, we attribute these differences to outliers in the small high-dosage sample for two reasons: (i) median pre-treatment differences are null, and (ii) the mean differences in the high-dosage group are driven by a single repository (*deepmind/mujoco*)—removing it removes all statistically significant differences.

Table A.3: **GitHub experiment: Pre-treatment balance tests of repository events.** Data comes from the 2023 GitHub Archive data, with 2023 event counts aggregated at the repository level for the period prior to the treatment window (May 12–20, 2023).

	(1)	(2)	(3)	(3)-(2)
	Full sample	Control repos	Treated repos	Pairwise t-test
Variable	Mean (SE)	Mean (SE)	Mean (SE)	Normalized diff
Stars	11.72 (3.43)	11.85 (3.95)	11.11 (5.63)	-0.01
Push events	30.61 (5.04)	29.45 (5.77)	36.42 (9.16)	0.06
Pull request events	11.62 (2.47)	10.51 (2.79)	17.16 (4.88)	0.12
Issues opened	2.10 (0.44)	1.86 (0.43)	3.31 (1.50)	0.12
Issues closed	1.87 (0.48)	1.62 (0.48)	3.13 (1.55)	0.12
Fork events	1.88 (0.37)	1.70 (0.38)	2.80 (1.10)	0.11
Release events	1.70 (0.19)	1.72 (0.22)	1.63 (0.40)	-0.02

Note: N = 582 (full sample), 485 (control), 97 (treated); pairwise test N = 582 (treated vs. control).

Table A.4: **GitHub experiment: Pre-treatment balance tests of repository events within low-dosage and high-dosage treatment groups.** Same as Table A.3, but separates treatment groups (low/high).

	(1)	(2)	(3)	(2)-(1)	(3)-(1)
	Control repos	Treated (low)	Treated (high)	Pairwise t-test	
Variable	Mean (SE)	Mean (SE)	Mean (SE)	Normalized diff	
Stars	11.85 (3.95)	4.62 (1.68)	30.88 (22.05)	-0.12	0.19
Push events	29.45 (5.77)	29.97 (10.71)	56.04 (17.35)	0.00	0.25
Pull request events	10.51 (2.79)	11.77 (4.43)	33.58 (14.14)	0.02	0.35
Issues opened	1.86 (0.43)	1.78 (0.94)	7.96 (5.31)	-0.01	0.31**
Issues closed	1.62 (0.48)	1.45 (0.90)	8.25 (5.61)	-0.02	0.32**
Fork events	1.70 (0.38)	1.22 (0.56)	7.63 (4.01)	-0.07	0.39**
Release events	1.72 (0.22)	1.88 (0.52)	0.88 (0.31)	0.03	-0.24

Note: N = 485 (control), 73 (treated, low), 24 (treated, high); pairwise tests N = 558 (low vs. control) and 509 (high vs. control). $^{*}p < .05$;$^{**}p < .01$;$^{***}p < .001$.

A.3: Manipulation Check

This section verifies that the treatment successfully increased GitHub stars for treated packages. Table A.5 reports differences in star counts between treated and control groups at the start and end of the treatment window.

Table A.5: **GitHub experiment: Manipulation checks.** Differences in GitHub star medians and means for the treatment and control groups. Star counts are cumulative, recording the total number of stars accrued to date. Column (1) reports differences in medians at the start of treatment (May 12). Column (2) reports differences in medians at the end of treatment (May 21). Columns (1)–(2) correspond to Figure 1. Columns (3)–(4) report differences in means, corresponding to Figure A.1.

	(1)	(2)	(3)	(4)
	Outcome variable is: GitHub stars			
	Difference in medians		Difference in means	
	On May 12	On May 21	On May 12	On May 21
Treatment (low)	0.00	20.00***	-0.27	18.43***
	(0.27)	(0.27)	(2.76)	(2.55)
	[−0.52, 0.52]	[19.48, 20.52]	[−5.69, 5.15]	[13.41, 23.44]
	< 1.000 >	< .000 >	< .923 >	< .000 >
Treatment (high)	0.00	69.00***	6.87	62.15***
	(0.87)	(0.87)	(6.81)	(4.57)
	[−1.70, 1.70]	[67.30, 70.70]	[−6.52, 20.25]	[53.16, 71.13]
	< 1.000 >	< .000 >	< .314 >	< .000 >
Constant	0.00	0.00	8.41***	8.69***
	(0.10)	(0.10)	(1.07)	(1.08)
	[−0.19, 0.19]	[−0.19, 0.19]	[6.31, 10.52]	[6.58, 10.81]
	< 1.000 >	< 1.000 >	< .000 >	< .000 >
Outcome median/mean	0.0	1.0	8.7	13.7
Package obs.	585	585	585	585
Day obs.	1	1	1	1
Package-day obs.	585	585	585	585

Note: Parentheses show standard errors clustered by packages, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

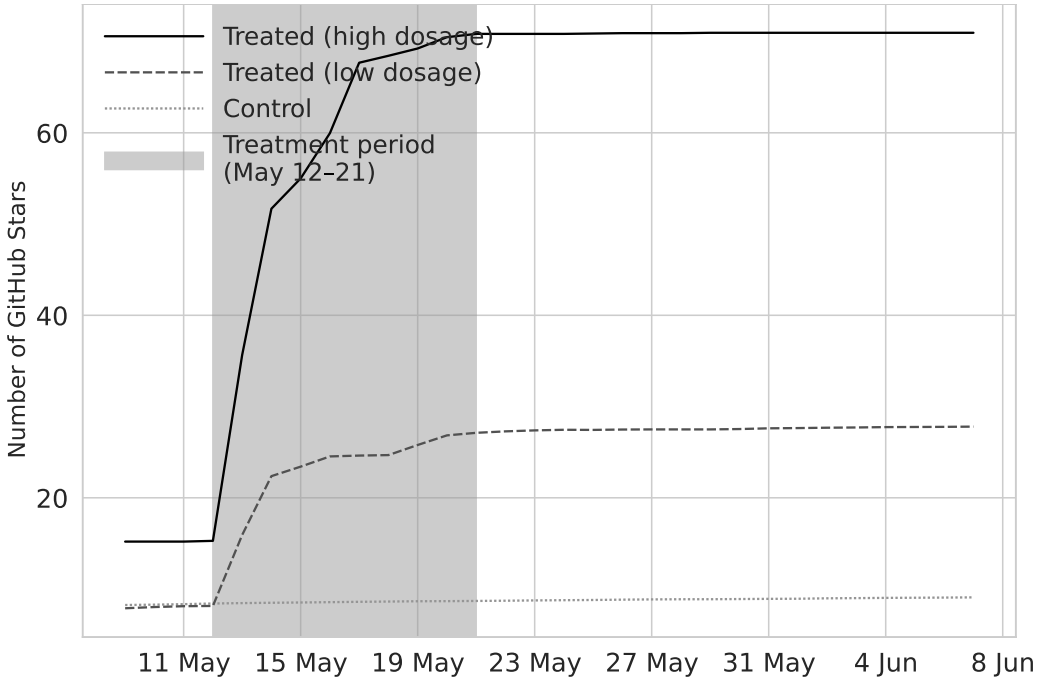

Figure A.1: **Manipulation Check: GitHub Stars for Treated vs. Control**. Same as Figure 1, except in means. The figure plots the mean number of stars on a particular day for the three groups: high-dosage treatment (market and network stars), low-dosage treatment (network stars), and control (no stars). In all, we plot data for 585 packages and 17,550 package days over the period May 9–June 7, 2023. The shaded vertical bar indicates the period during which the treatment was being applied. Table A.5 presents estimates of the impact of treatment on stars.

A.4: Alternate Estimands of the Treatment Effect

This section reports the ITT estimates using two alternate estimands—differences in medians (Section A.4.1) and differences in means (Section A.4.2). Section A.4.2 reports differences in means—while there appears to be a break in trend in Figure A.2, the within-group variance is large, driven in part by extreme outliers (Figures A.3 to A.4) which we identified from the data (Table A.8). We do not observe a similar trend break for the medians. We also examine heterogeneity by package complexity (Section A.4.3), using two proxies: repository size in MB and README documentation length, each split at the median into above- and below-median groups (Tables A.9 and A.10).

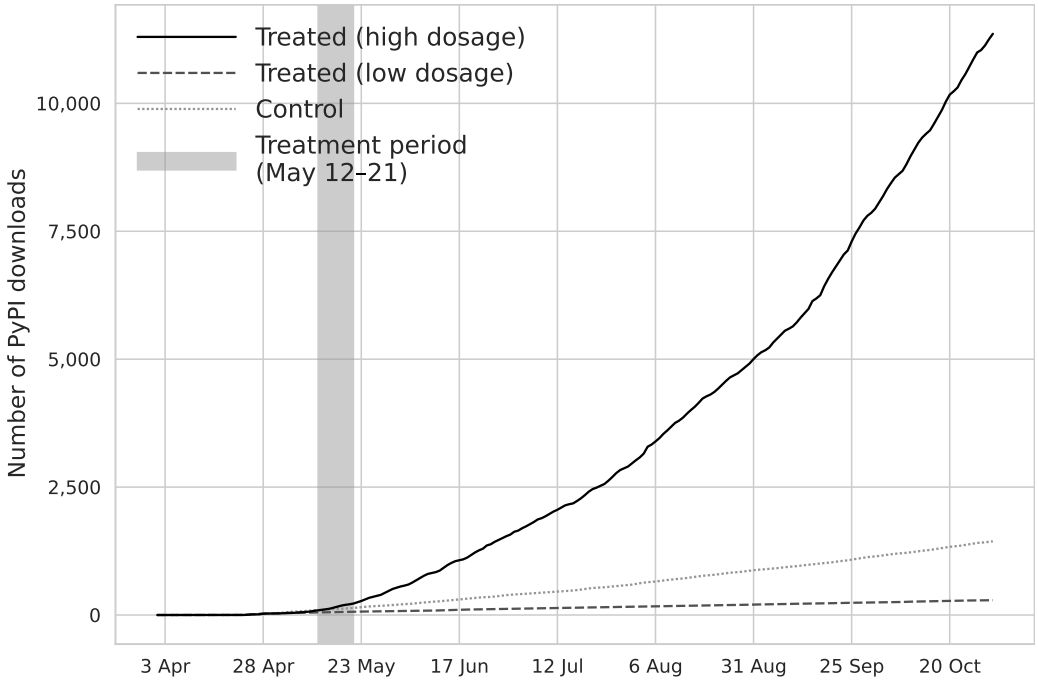

Figure A.2: **Mean PyPI downloads for Treated vs. Control in GitHub Experiment.** Same as Figure 2, except for means. The figure plots the mean of cumulative downloads for each of the three groups. Each point is a day averaged within the group for 622 packages and 133,108 package days over the sample period April–October 2023. Treatment is distinguished by low and high dosages (see Section 3.2). The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table B.2). See also Figures A.3 to A.4 for the time series of individual packages. Table A.7 reports the estimates of differences in means.

A.4.1: Differences in Medians

Table A.6: **GitHub Experiment Results: ITT estimates for median downloads.** Findings correspond to Figure 2. Columns (1)–(5) report snapshots of the difference in medians at various dates. Column (6) reports post-treatment differences in medians over the full post-treatment period of 165 days (May 20–October 31, 2023), allowing for heterogeneous treatment effects through a linear time trend.

	(1)	(2)	(3)	(4)	(5)	(6)
	Outcome variable is: PyPI downloads					
	On Jun 20	On Jul 20	On Aug 20	On Sep 20	On Oct 20	Full post period
Treatment (low)	4.0	9.0	14.0	7.0	13.0	3.7
	(8.7)	(11.7)	(13.8)	(15.7)	(24.3)	(3.7)
	[−13.2, 21.2]	[−14.1, 32.1]	[−13.1, 41.1]	[−23.8, 37.8]	[−34.8, 60.8]	[−3.6, 11.0]
	<.647>	<.444>	<.311>	<.655>	<.593>	<.323>
Treatment (high)	5.0	8.0	-2.0	-5.0	10.0	6.8
	(23.9)	(23.9)	(25.4)	(24.8)	(30.7)	(8.2)
	[−41.8, 51.8]	[−38.9, 54.9]	[−51.8, 47.8]	[−53.6, 43.6]	[−50.2, 70.2]	[−9.3, 22.9]
	<.834>	<.738>	<.937>	<.840>	<.744>	<.407>
Linear trend						0.3***
						(0.0)
						[0.3, 0.4]
						<.000>
Treatment (low) × Linear trend						0.1
						(0.2)
						[−0.4, 0.5]
						<.737>
Treatment (high) × Linear trend						-0.0
						(0.2)
						[−0.5, 0.4]
						<.911>
Constant	44.0***	53.0***	65.0***	74.0***	84.0***	31.4***
	(2.5)	(2.8)	(4.1)	(5.4)	(6.9)	(1.8)
	[39.1, 48.9]	[47.5, 58.5]	[56.9, 73.1]	[63.4, 84.6]	[70.4, 97.6]	[27.9, 34.9]
	<.000>	<.000>	<.000>	<.000>	<.000>	<.000>
Median of outcome	44.0	54.0	65.0	74.0	84.0	59.0
Package obs.	622	622	622	622	622	622
Day obs.	1	1	1	1	1	165
Package-day obs.	622	622	622	622	622	102,630

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. $^{*}p < .05$;$^{**}p < .01$;$^{***}p < .001$.

A.4.2: Differences in Means

While there appears to be a break in trend in Figure A.2, we note three additional time series behaviors. First, the within-group variance is large, as will be evident in the estimates (Table A.7). Second, this, at least in part, can be explained by a few extreme outliers (Figures A.3 to A.4). Third, and relatedly, we do not observe a similar trend break for the medians (Figure 2).

Table A.7 reports the ITT estimates. Approximately one month after intervention, on June 20, the low-dosage group's mean download tally was 219 lower than the control group (p = .080), while the high-dosage group's download tally was higher but statistically indistinguishable from the control group (p = .449, Table A.7). Three months after intervention (on August 20), the low-dosage group's mean download tally was 590 lower than the control group (p = .070), while the high-dosage group's download tally was 3,532 higher than the control group (p = .388), with the standard errors of the estimates always in the same order of magnitude as the estimates (Table A.7). We also estimated a model that allows the treatment effect to vary over time (column (6) of Table A.7). Neither the low-dosage nor high-dosage group had a trend different from the control group. If anything, the low-dosage treatment group had a weaker trend than the control group (p = .066).

Table A.8 lists four outlier packages, the highest trajectories in Figure A.3, together with their GitHub repository URLs and treatment assignment. First, *pyaesm* is a lightweight implementation of the Advanced Encryption Standard (AES) encryption algorithm, requiring no external dependencies beyond Python's built-in modules. While the PyPI package was released on April 28, 2023, the underlying source code in the GitHub repository had existed for many years prior to the PyPI release. Second, *dghs-imgutils* implements "anime-style image data processing" using pretrained models, including those for clustering, object detection, LineArt generation, and character extraction. The repository includes extensive documentation, visual examples, and demonstration code in its README, which may contribute to its visibility, engagement, and uptake. Third, *salt-analytics-framework* is an extension to the Salt automation and orchestration system that adds support to "collect, process and forward analytics/metrics data". Salt is itself an established open-source infrastructure automation and configuration management system (with 15k+ stars and 82k/month downloads). The observed growth in downloads of *salt-analytics-framework*, therefore, very likely reflects ecosystem spillovers from Salt. Finally, Firebase is a Google-managed platform that allows developers to build web and mobile applications: *firebase-functions* is its official Python SDK for integrating with Google's Firebase and Cloud Functions infrastructure.

Of the four, three were assigned to control (*pyaesm*, *dghs-imgutils*, *salt-analytics-framework*), making it unlikely that they were affected by our treatment. Two (*salt-analytics-framework*, *firebase-functions*) have uptake in usage (downloads) that very likely reflect positive spillovers from large established ecosystems. Only one (*firebase-*

functions) was assigned to our treatment group, but it is unlikely to have benefited from our manipulation. Instead, the uptick in downloads coinciding with our treatment period is more likely to have arisen from Google's announcement of Python support for Firebase Cloud Functions at Google I/O on May 10 (2 days prior to our treatment implementation).[15]

Table A.7: **GitHub Experiment Results: ITT estimates for means downloads.** Similar to Table A.6, but for differences in means. Corresponds to Figure A.2. Columns (1)–(5) report snapshots of the difference in mean at various dates. Column (6) reports post-treatment differences in means over the full post-treatment period of 165 days (May 20–October 31, 2023), allowing for heterogeneous treatment effects through a linear time trend.

	(1)	(2)	(3)	(4)	(5)	(6)
	Outcome variable is: PyPI downloads					
	On Jun 20	On Jul 20	On Aug 20	On Sep 20	On Oct 20	Full post period
Treatment (low)	-219.9	-374.2	-590.4	-800.2	-1,054.7	-6.3
	(125.3)	(215.4)	(325.1)	(429.4)	(571.0)	(38.7)
	[−466.0, 26.3]	[−797.2, 48.7]	[−1,228.8, 48.0]	[−1,643.5, 43.1]	[−2,176.1, 66.7]	[−82.2, 69.7]
	<.080>	<.083>	<.070>	<.063>	<.065>	<.872>
Treatment (high)	846.6	1,892.1	3,531.9	5,665.3	8,836.4	-1,147.5
	(1,117.3)	(2,282.7)	(4,092.7)	(6,462.5)	(9,975.0)	(1,095.6)
	[−1,347.6, 3,040.8]	[−2,590.7, 6,374.9]	[−4,505.3, 11,569.1]	[−7,025.8, 18,356.3]	[−10,752.6, 28,425.3]	[−3,299.1, 1,004.0]
	<.449>	<.407>	<.388>	<.381>	<.376>	<.295>
Linear trend						8.0*
						(3.6)
						[1.0, 15.0]
						<.025>
Treatment (low) × Linear trend						-6.6
						(3.6)
						[−13.6, 0.4]
						<.066>
Treatment (high) × Linear trend						58.3
						(62.5)
						[−64.4, 181.0]
						<.351>
Constant	326.9**	519.0*	776.9*	1,029.7*	1,331.6*	67.5
	(122.8)	(212.6)	(321.7)	(425.2)	(566.7)	(35.5)
	[85.7, 568.1]	[101.5, 936.4]	[145.2, 1,408.7]	[194.7, 1,864.8]	[218.6, 2,444.5]	[−2.2, 137.3]
	<.008>	<.015>	<.016>	<.016>	<.019>	<.058>
R^2	0.00445	0.00648	0.00881	0.0114	0.0137	0.0140
Mean of outcome	334.4	549.9	847.7	1,161.0	1,559.5	801.2
Package obs.	622	622	622	622	622	622
Day obs.	1	1	1	1	1	165
Package-day obs.	622	622	622	622	622	102,630

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

15. See, for example, https://firebase.blog/posts/2023/05/whats-new-at-google-io/ and https://techcrunch.com/2023/05/10/googles-firebase-gets-ai-extensions-opens-up-its-marketplace/.

Table A.8: **Outlier Packages with Anomalously High Download Counts.** Outliers with more than 100,000 total downloads (see Figure A.3). Total downloads measured as of the end of the sample period on Oct 31, 2023.

	Package	GitHub Repository URL	Assignment	Total Downloads
1	pyaesm	https://github.com/ricmoo/pyaes	Control	123,198
2	dghs-imgutils	https://github.com/deepghs/imgutils	Control	118,853
3	salt-analytics-framework	https://github.com/saltstack/salt-analytics-framework	Control	580,519
4	firebase-functions	https://github.com/firebase/firebase-functions-python	Treated (High)	525,229

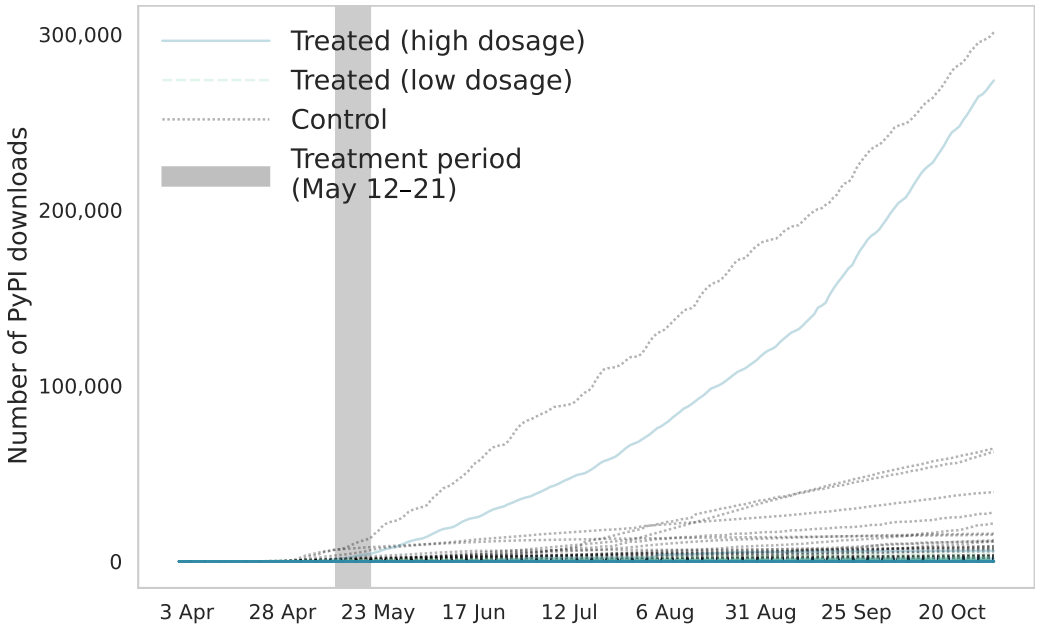

Figure A.3: **Individual PyPI downloads for Treated vs Control.** Similar to Figure A.2, but with the time series of the individual packages. The four packages with the highest download trajectories (> 100,000 total downloads) are identified and discussed in Table A.8. See Figure A.4 for the same figure with the two most extreme outliers removed.

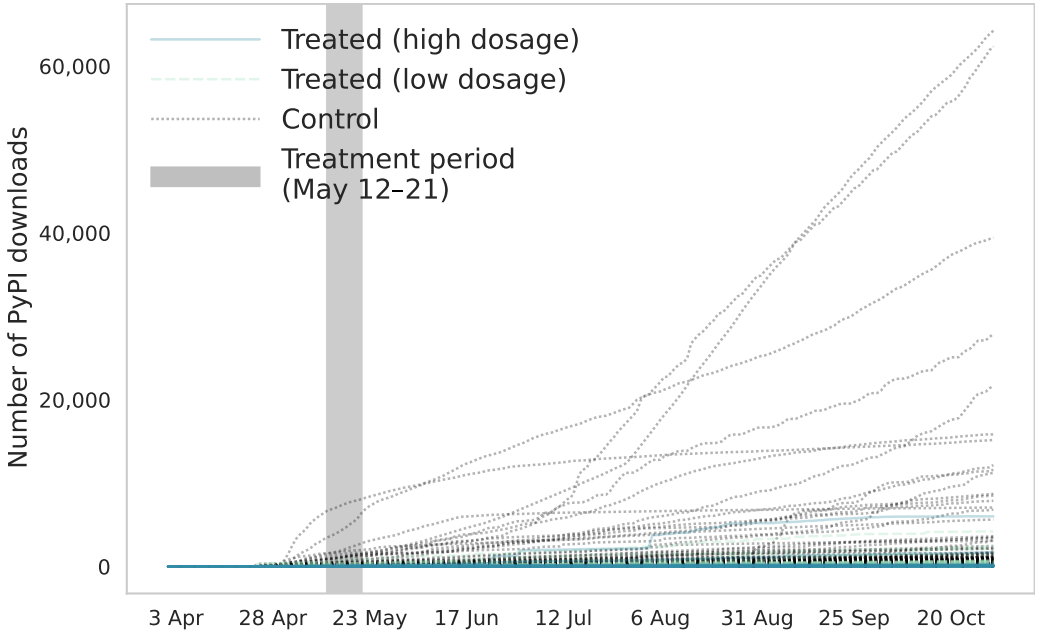

Figure A.4: **Individual PyPI downloads for treated vs control.** Similar to Figure A.3, but without the top two extreme time series.

A.4.3: Heterogeneity by Package Complexity

Table A.9: **GitHub Experiment Results: ITT estimates by codebase size.** Similar to Table A.6, but modeled to allow for differential effects by repository complexity. The 'Large repository' moderator indicates packages with an above-median size (total size of all files in the GitHub repository, including the codebase, in MB). Columns (1)–(5) report differences in medians at monthly snapshots.

	(1)	(2)	(3)	(4)	(5)
	Outcome variable is: PyPI downloads				
	On Jun 20	On Jul 20	On Aug 20	On Sep 20	On Oct 20
Treatment (low)	-4.0	0.0	-6.0	-12.0	-13.0
	(5.8)	(8.3)	(10.9)	(13.6)	(15.4)
	[−15.4, 7.4]	[−16.3, 16.3]	[−27.4, 15.4]	[−38.8, 14.8]	[−43.3, 17.3]
	<.493>	<1.000>	<.582>	<.379>	<.400>
Treatment (high)	19.0	14.0	4.0	-6.0	-9.0
	(29.4)	(33.3)	(47.0)	(46.5)	(46.6)
	[−38.7, 76.7]	[−51.4, 79.4]	[−88.2, 96.2]	[−97.3, 85.3]	[−100.6, 82.6]
	<.518>	<.674>	<.932>	<.897>	<.847>
Large repository	6.0	7.0	10.0	6.0	10.0
	(4.6)	(6.2)	(8.2)	(10.6)	(12.9)
	[−2.9, 14.9]	[−5.3, 19.3]	[−6.2, 26.2]	[−14.9, 26.9]	[−15.3, 35.3]
	<.188>	<.263>	<.225>	<.573>	<.439>
Treatment (low)	32.0	40.0	48.0	58.0	58.0
× Large repository	(22.7)	(27.6)	(28.9)	(32.9)	(34.9)
	[−12.5, 76.5]	[−14.3, 94.3]	[−8.7, 104.7]	[−6.6, 122.6]	[−10.5, 126.5]
	<.158>	<.148>	<.097>	<.078>	<.097>
Treatment (high)	-25.0	-13.0	0.0	28.0	54.0
× Large repository	(45.2)	(47.5)	(64.4)	(62.6)	(65.9)
	[−113.7, 63.7]	[−106.4, 80.4]	[−126.4, 126.4]	[−94.9, 150.9]	[−75.5, 183.5]
	<.580>	<.785>	<1.000>	<.655>	<.413>
Constant	40.0***	47.0***	57.0***	69.0***	76.0***
	(3.2)	(3.9)	(4.8)	(6.1)	(7.3)
	[33.8, 46.2]	[39.4, 54.6]	[47.5, 66.5]	[57.1, 80.9]	[61.7, 90.3]
	<.000>	<.000>	<.000>	<.000>	<.000>
Median of outcome	44	53	63	71	82
Package obs.	585	585	585	585	585
Day obs.	1	1	1	1	1
Package-day obs.	585	585	585	585	585

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

Table A.10: **GitHub Experiment Results: ITT estimates by README documentation length.** Similar to Table A.6, but modeled to allow for differential effects by repository complexity. The 'Long readme' moderator indicates packages with an above-median README documentation length (cleaned for HTML markup; raw text is similarly null). Columns (1)–(5) report differences in medians at monthly snapshots.

	(1)	(2)	(3)	(4)	(5)
	Outcome variable is: PyPI downloads				
	On Jun 20	On Jul 20	On Aug 20	On Sep 20	On Oct 20
Treatment (low)	-4.0	0.0	2.0	-8.0	-13.0
	(8.0)	(15.3)	(16.5)	(19.7)	(23.9)
	[-19.6, 11.6]	[-30.1, 30.1]	[-30.4, 34.4]	[-46.7, 30.7]	[-59.8, 33.8]
	<.615>	<1.000>	<.904>	<.685>	<.586>
Treatment (high)	29.0	38.0	55.0	44.0	39.0
	(26.8)	(28.9)	(32.2)	(34.4)	(22.6)
	[-23.7, 81.7]	[-18.8, 94.8]	[-8.3, 118.3]	[-23.6, 111.6]	[-5.4, 83.4]
	<.280>	<.189>	<.089>	<.202>	<.085>
Long readme	0.0	5.0	13.0	15.0	15.0
	(5.0)	(6.0)	(7.6)	(10.1)	(12.8)
	[-9.8, 9.8]	[-6.7, 16.7]	[-1.9, 27.9]	[-4.9, 34.9]	[-10.1, 40.1]
	<1.000>	<.403>	<.087>	<.139>	<.242>
Treatment (low)	17.0	14.0	22.0	24.0	56.0
× Long readme	(12.8)	(22.1)	(25.0)	(29.2)	(43.6)
	[-8.1, 42.1]	[-29.5, 57.5]	[-27.2, 71.2]	[-33.4, 81.4]	[-29.6, 141.6]
	<.185>	<.527>	<.380>	<.412>	<.199>
Treatment (high)	-33.0	-37.0	-63.0	-57.0	-48.0
× Long readme	(30.4)	(29.6)	(34.2)	(41.7)	(46.7)
	[-92.7, 26.7]	[-95.1, 21.1]	[-130.1, 4.1]	[-138.9, 24.9]	[-139.7, 43.7]
	<.278>	<.211>	<.066>	<.172>	<.304>
Constant	44.0***	49.0***	56.0***	67.0***	76.0***
	(4.3)	(5.0)	(5.4)	(6.6)	(9.4)
	[35.5, 52.5]	[39.2, 58.8]	[45.4, 66.6]	[54.0, 80.0]	[57.5, 94.5]
	<.000>	<.000>	<.000>	<.000>	<.000>
Median of outcome	44	54	65	74	84
Package obs.	622	622	622	622	622
Day obs.	1	1	1	1	1
Package-day obs.	622	622	622	622	622

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. $^{*}p < .05$;$^{**}p < .01$;$^{***}p < .001$.

A.5: GitHub Events as Outcomes

In this section, we use the seven GitHub activity metrics (stars, forks, pushes, pull requests, opened issues, closed issues, and releases) as outcomes. Specifically, we estimate whether treated groups have a difference in those metrics at a post-treatment snapshot (July 1) and whether there is a linear trend across the entire post-treatment period (to the end of 2023).

For the GitHub experiment, we find no post-treatment difference in any metric, except that treated groups have more stars, which is another confirmation that manipulation was in place (Table A.11 and Table A.12). We find similarly null results when we look at the post-treatment snapshot on December 31. Unlike Table A.6, we do not estimate median differences since the median for most GitHub activity metrics is zero in both groups throughout the observation window (causing the estimator to degenerate).

Table A.11: **GitHub Experiment: Post-treatment differences in cumulative GitHub repository event activity—stars, push events, and pull requests (ITT).** Odd-numbered columns report differences in means at a snapshot approximately six weeks after intervention ended (1 Jul 2023). Even-numbered columns report post-treatment differences in means over the full post-treatment period, allowing for heterogeneous treatment effects through a linear time trend.

	(1)	(2)	(3)	(4)	(5)	(6)
	Stars		Push events		Pull requests	
	Jul 1	Full	Jul 1	Full	Jul 1	Full
Treatment group	29.1**	29.8**	8.7	6.2	7.8	5.9
	(10.2)	(9.2)	(13.7)	(11.2)	(7.6)	(6.1)
	[9.1, 49.1]	[11.8, 47.8]	[−18.2, 35.5]	[−15.7, 28.2]	[−7.1, 22.7]	[−6.0, 17.9]
	< .004 >	< .001 >	< .527 >	< .577 >	< .302 >	< .329 >
Linear trend		0.1***		0.1***		0.1***
		(0.0)		(0.0)		(0.0)
		[0.0, 0.1]		[0.1, 0.2]		[0.0, 0.1]
		< .000 >		< .000 >		< .000 >
Treatment group × Linear trend		-0.0		0.1		0.0
		(0.0)		(0.1)		(0.0)
		[−0.1, 0.1]		[−0.1, 0.2]		[−0.0, 0.1]
		< .850 >		< .381 >		< .246 >
Constant	18.1**	14.8**	39.8***	33.9***	15.4***	12.5***
	(5.6)	(4.8)	(7.2)	(6.3)	(3.6)	(3.1)
	[7.2, 29.0]	[5.4, 24.2]	[25.6, 53.9]	[21.5, 46.3]	[8.3, 22.4]	[6.4, 18.6]
	< .001 >	< .002 >	< .000 >	< .000 >	< .000 >	< .000 >
Outcome median/mean	1	1	12	13	0	0
Repository obs.	582	582	582	582	582	582
Day obs.	1	42	1	42	1	42
Repository-day obs.	582	130,950	582	130,950	582	130,950

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

Table A.12: **GitHub Experiment: Post-treatment differences in cumulative GitHub repository event activity—issues opened, issues closed, fork events, and release events (ITT).** Similar to Table A.11, but for issues opened/closed, fork events, and release events.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
	Issues opened		Issues closed		Fork events		Release events	
	Jul 1	Full	Jul 1	Full	Jul 1	Full	Jul 1	Full
Treatment group	1.9	1.4	1.9	1.6	1.4	1.3	-0.4	-0.2
	(2.3)	(2.0)	(2.2)	(1.9)	(1.6)	(1.4)	(0.5)	(0.5)
	[−2.6, 6.4]	[−2.5, 5.2]	[−2.5, 6.2]	[−2.2, 5.4]	[−1.8, 4.6]	[−1.5, 4.1]	[−1.4, 0.6]	[−1.2, 0.7]
	< .410 >	< .478 >	< .404 >	< .416 >	< .380 >	< .351 >	< .447 >	< .621 >
Linear trend		0.0***		0.0***		0.0***		0.0***
		(0.0)		(0.0)		(0.0)		(0.0)
		[0.0, 0.0]		[0.0, 0.0]		[0.0, 0.0]		[0.0, 0.0]
		< .000 >		< .000 >		< .000 >		< .000 >
Treatment group		0.0		0.0		0.0		-0.0
× Linear trend		(0.0)		(0.0)		(0.0)		(0.0)
		[−0.0, 0.0]		[−0.0, 0.0]		[−0.0, 0.0]		[−0.0, 0.0]
		< .319 >		< .427 >		< .373 >		< .124 >
Constant	2.8***	2.2***	2.3***	1.9***	2.6***	2.1***	2.3***	2.0***
	(0.6)	(0.5)	(0.6)	(0.5)	(0.6)	(0.5)	(0.3)	(0.2)
	[1.7, 4.0]	[1.3, 3.2]	[1.1, 3.6]	[0.8, 2.9]	[1.5, 3.8]	[1.2, 3.0]	[1.7, 2.9]	[1.5, 2.5]
	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >
Outcome median/mean	0	0	0	0	0	0	0	0
Repository obs.	582	582	582	582	582	582	582	582
Day obs.	1	42	1	42	1	42	1	42
Repository-day obs.	582	130,950	582	130,950	582	130,950	582	130,950

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. $^{*}p < .05$;** $p < .01$;*** $p < .001$.

Table A.13: **GitHub Experiment: Post-treatment differences in cumulative GitHub repository event activity—stars, push events, and pull requests (ITT) (low-dosage and high-dosage treatment groups vs. control).** Odd-numbered columns report differences in means at a snapshot approximately six weeks after intervention ended (1 Jul 2023). Even-numbered columns report post-treatment differences in means over the full post-treatment period, allowing for heterogeneous treatment effects through a linear time trend.

	(1)	(2)	(3)	(4)	(5)	(6)
	Stars		Push events		Pull requests	
	Jul 1	Full	Jul 1	Full	Jul 1	Full
Treatment (low)	9.5	10.7*	0.1	-1.9	1.0	0.0
	(6.2)	(5.2)	(15.2)	(12.1)	(7.2)	(5.7)
	[−2.6, 21.6]	[0.5, 21.0]	[−29.7, 29.9]	[−25.8, 21.9]	[−13.1, 15.2]	[−11.1, 11.1]
	< .124 >	< .041 >	< .996 >	< .874 >	< .884 >	< .999 >
Treatment (high)	88.7**	87.8**	34.8	31.0	28.5	24.0
	(31.8)	(28.3)	(25.0)	(20.0)	(19.6)	(15.3)
	[26.2, 151.1]	[32.3, 143.3]	[−14.3, 83.8]	[−8.2, 70.3]	[−10.1, 67.0]	[−6.0, 54.0]
	< .005 >	< .002 >	< .164 >	< .121 >	< .147 >	< .116 >
Linear trend		0.1***		0.1***		0.1***
		(0.0)		(0.0)		(0.0)
		[0.0, 0.1]		[0.1, 0.2]		[0.0, 0.1]
		< .000 >		< .000 >		< .000 >
Treatment (low)		-0.0		0.0		0.0
× Linear trend		(0.0)		(0.1)		(0.0)
		[−0.1, 0.0]		[−0.1, 0.2]		[−0.0, 0.1]
		< .354 >		< .644 >		< .536 >
Treatment (high)		0.0		0.1		0.1
× Linear trend		(0.1)		(0.1)		(0.1)
		[−0.1, 0.2]		[−0.1, 0.4]		[−0.1, 0.3]
		< .494 >		< .298 >		< .241 >
Constant	18.1**	14.8**	39.8***	33.9***	15.4***	12.5***
	(5.6)	(4.8)	(7.2)	(6.3)	(3.6)	(3.1)
	[7.2, 29.0]	[5.4, 24.2]	[25.6, 53.9]	[21.5, 46.3]	[8.3, 22.4]	[6.4, 18.6]
	< .001 >	< .002 >	< .000 >	< .000 >	< .000 >	< .000 >
Outcome median/mean	1	1	12	13	0	0
Repository obs.	582	582	582	582	582	582
Day obs.	1	42	1	42	1	42
Repository-day obs.	582	130,950	582	130,950	582	130,950

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

Table A.14: **GitHub Experiment: Post-treatment differences in cumulative GitHub repository event activity—issues opened, issues closed, fork events, and release events (ITT) (low-dosage and high-dosage treatment groups vs. control).** Similar to Table A.13, but for issues opened/closed, fork events, and release events.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
	Issues opened		Issues closed		Fork events		Release events	
	Jul 1	Full	Jul 1	Full	Jul 1	Full	Jul 1	Full
Treatment (low)	-0.5	-0.8	-0.4	-0.6	-0.7	-0.6	-0.2	-0.0
	(1.4)	(0.9)	(1.4)	(0.9)	(1.0)	(0.8)	(0.6)	(0.6)
	[−3.2, 2.3]	[−2.7, 1.0]	[−3.1, 2.3]	[−2.4, 1.2]	[−2.6, 1.3]	[−2.1, 0.9]	[−1.4, 1.1]	[−1.2, 1.1]
	< .748 >	< .392 >	< .758 >	< .509 >	< .490 >	< .416 >	< .783 >	< .949 >
Treatment (high)	9.0	8.1	8.8	8.3	7.8	7.2	-1.1*	-0.9
	(8.2)	(7.1)	(7.9)	(7.1)	(5.6)	(4.9)	(0.5)	(0.4)
	[−7.1, 25.2]	[−5.9, 22.1]	[−6.7, 24.3]	[−5.6, 22.1]	[−3.2, 18.9]	[−2.4, 16.8]	[−2.1, −0.1]	[−1.7, 0.0]
	< .272 >	< .257 >	< .264 >	< .242 >	< .164 >	< .140 >	< .033 >	< .055 >
Linear trend		0.0***		0.0***		0.0***		0.0***
		(0.0)		(0.0)		(0.0)		(0.0)
		[0.0, 0.0]		[0.0, 0.0]		[0.0, 0.0]		[0.0, 0.0]
		< .000 >		< .000 >		< .000 >		< .000 >
Treatment (low)		0.0		0.0		0.0		-0.0
× Linear trend		(0.0)		(0.0)		(0.0)		(0.0)
		[−0.0, 0.0]		[−0.0, 0.0]		[−0.0, 0.0]		[−0.0, 0.0]
		< .545 >		< .719 >		< .979 >		< .192 >
Treatment (high)		0.0		0.0		0.0		-0.0
× Linear trend		(0.0)		(0.0)		(0.0)		(0.0)
		[−0.0, 0.1]		[−0.0, 0.1]		[−0.0, 0.1]		[−0.0, 0.0]
		< .284 >		< .294 >		< .180 >		< .107 >
Constant	2.8***	2.2***	2.3***	1.9***	2.6***	2.1***	2.3***	2.0***
	(0.6)	(0.5)	(0.6)	(0.5)	(0.6)	(0.5)	(0.3)	(0.2)
	[1.7, 4.0]	[1.3, 3.2]	[1.1, 3.6]	[0.8, 2.9]	[1.5, 3.8]	[1.2, 3.0]	[1.7, 2.9]	[1.5, 2.5]
	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >
Outcome median/mean	0	0	0	0	0	0	0	0
Repository obs.	582	582	582	582	582	582	582	582
Day obs.	1	42	1	42	1	42	1	42
Repository-day obs.	582	130,950	582	130,950	582	130,950	582	130,950

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

Appendix B: PyPI Experiment

B.1: Balance Tests Using GitHub Archive Events

In this section, we constructed pre-treatment balance tables using historical GitHub event data from the GitHub Archive Project. We aggregated cumulative event counts for each repository from January 1, 2023, to the start of each experiment's treatment period (June 3 for the PyPI experiment) across seven GitHub activity metrics: stars (WatchEvent), forks, pushes, pull requests, issues opened, issues closed, and releases.

For the PyPI experiment, we first retrieved GitHub URLs for the 23,965 packages in the sample, finding similar rates of recovery across groups (75.3% vs. 74.3% for treated vs. control, N = 17,872 with linked repositories). We then mapped their GitHub repositories to GHArchive events as above. We observe no pre-treatment difference between the treated and control groups on any of the seven metrics (Table B.1).

Table B.1: **PyPI experiment: Balance tests of repository events.** Data are from the 2023 GitHub Archive data, aggregated at the repository level for the period prior to the treatment window (Jun 3–8, 2023). N = 17,872 (full sample), 14,241 (control), 3,631 (treated); pairwise test uses N = 17,872 (treated vs. control).

Variable	(1) Full sample Mean/(SE)	(2) Control repos Mean/(SE)	(3) Treated repos Mean/(SE)	(3)-(2) Pairwise t-test Normalized diffs
Stars	24.41	24.51	24.04	-0.00
	(4.45)	(5.48)	(4.23)	
Push events	44.30	43.03	49.30	0.02
	(2.50)	(2.74)	(6.05)	
Pull request events	31.26	29.67	37.48	0.03
	(1.74)	(1.87)	(4.42)	
Issues opened	6.49	6.27	7.34	0.01
	(0.61)	(0.72)	(0.98)	
Issues closed	5.35	5.16	6.11	0.02
	(0.45)	(0.53)	(0.81)	
Fork events	7.20	7.11	7.53	0.00
	(1.17)	(1.44)	(1.05)	
Release events	1.29	1.25	1.47	0.02
	(0.10)	(0.11)	(0.23)	

Note: Parentheses show standard errors, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *p < .05;** p < .01;*** p < .001.

B.2: Human vs. Bot Downloads

PyPI's Linehaul project logs each download with the installer type. This section classifies installers as human or bot based on their known function. "Human installers" are package managers invoked directly by users (e.g., pip, conda, poetry). "Bot installers" are typically caching mirrors or automated crawlers used for distributional and security purposes.

Table B.2: **Classification of human vs bot downloads by package installer.**

Installer Name	Type
pip	Human
Browser	Bot
Bandersnatch	Bot
setuptools	Human
Nexus	Human
requests	Bot
devpi	Bot
pdm	Human
Homebrew	Human
Artifactory	Human
OS	Human
Bazel	Human
pex	Human
conda	Human
chaquopy	Human

B.3: Alternate Estimands of the Treatment Effect

This section reports ITT estimates using two alternate estimands—differences in medians (Section B.3.1) and differences in means (Section B.3.2).

B.3.1: Differences in Medians

Table B.3: **PyPI Experiment Results: ITT estimates for differences in cumulative medians.** Columns (1)–(4) report differences in medians at monthly snapshots 1–4 months after the end of the treatment period (8 Jun 2023). Column (5) estimates differences in medians over the full post-treatment period (Jun 8–Oct 31, 2023), allowing for heterogeneous treatment effects through a linear time trend over 145 days. Corresponds to Figure 3.

	(1)	(2)	(3)	(4)	(5)
	Outcome variable is: Cumulative PyPI downloads				
	Jul 8	Aug 8	Sep 8	Oct 8	Full post period
Treatment group	75.0***	74.0***	72.0***	74.0***	75.2***
	(3.1)	(4.1)	(5.1)	(5.9)	(2.2)
	[68.9, 81.1]	[66.0, 82.0]	[61.9, 82.1]	[62.4, 85.6]	[70.9, 79.5]
	<.000>	<.000>	<.000>	<.000>	<.000>
Linear trend					0.8***
					(0.0)
					[0.8, 0.8]
					<.000>
Treatment group × Linear trend					-0.0
					(0.0)
					[-0.1, 0.1]
					<.657>
Constant	81.0***	104.0***	131.0***	154.0***	55.1***
	(1.4)	(1.8)	(2.3)	(2.6)	(0.9)
	[78.3, 83.7]	[100.4, 107.6]	[126.5, 135.5]	[149.0, 159.0]	[53.3, 56.9]
	<.000>	<.000>	<.000>	<.000>	<.000>
Median of outcome	109.0	129.0	153.0	174.0	135.0
Package obs.	23,965	23,965	23,965	23,965	23,965
Day obs.	1	1	1	1	146
Package-day obs.	23,965	23,965	23,965	23,965	3,498,890

Note: Parentheses show standard errors clustered by packages, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

B.3.2: Differences in Means

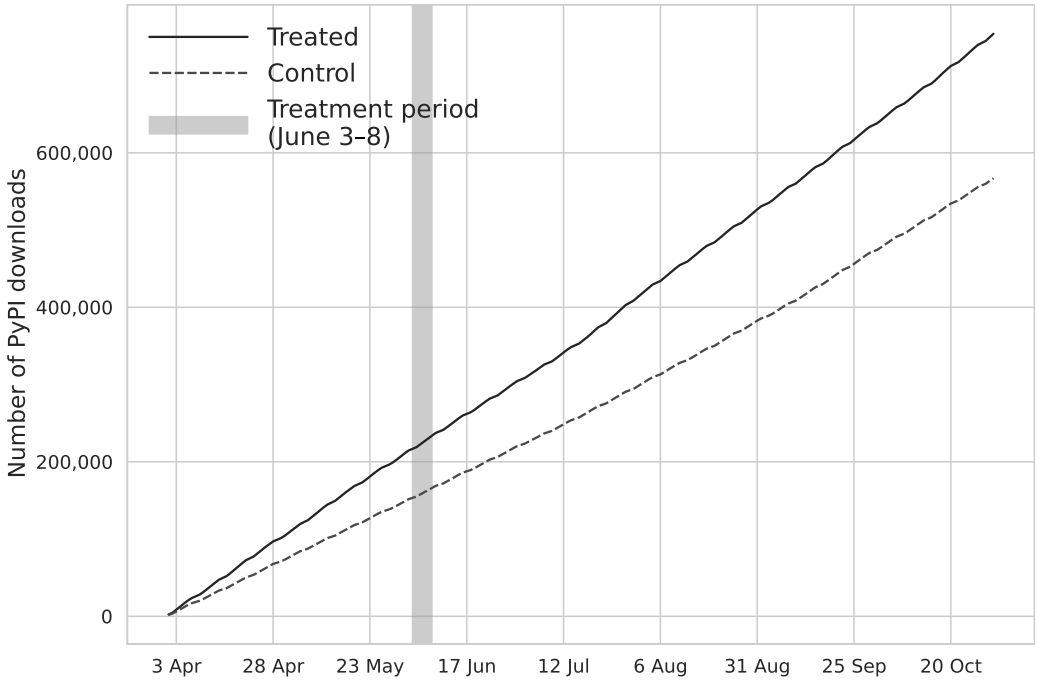

Figure B.1: **Mean PyPI downloads for Treated vs. Control in PyPI Experiment.** Same as Figure 3, except in means. The figure shows trends in mean daily downloads for the treated packages (*N* = 4,823) and control group packages (*N* = 19,142) for 5,128,510 package-day observations over the sample period April–October 2023. The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table B.2). Table B.4 reports the estimates in the differences in means.

Table B.4: **PyPI Experiment Results: ITT estimates for differences in cumulative means.** Columns (1)–(4) report differences in means at monthly snapshots 1–4 months after the end of the treatment period (8 Jun 2023). Column (5) estimates differences in means over the full post-treatment period (Jun 8–Oct 31, 2023), allowing for heterogeneous treatment effects through a linear time trend over 145 days. Corresponds to Figure B.1.

	(1)	(2)	(3)	(4)	(5)
	Outcome variable is: Cumulative PyPI downloads				
	Jul 8	Aug 8	Sep 8	Oct 8	Full post period
Treatment group	89,549.9 (167,175.0) [−238,123.5, 417,223.4] <.592>	123,138.4 (220,538.6) [−309,131.2, 555,408.0] <.577>	150,802.9 (273,236.7) [−384,758.2, 686,364.0] <.581>	168,361.2 (325,097.0) [−468,849.4, 805,571.8] <.605>	67,818.0 (117,656.0) [−162,795.1, 298,431.0] <.564>
Linear trend					2,770.5*** (657.5) [1,481.7, 4,059.2] <.000>
Treatment group × Linear trend					849.0 (1,715.4) [−2,513.3, 4,211.4] <.621>
Constant	238,386.1*** (53,805.5) [132,924.0, 343,848.2] <.000>	319,222.0*** (72,422.3) [177,269.8, 461,174.2] <.000>	404,842.2*** (93,003.3) [222,549.9, 587,134.5] <.000>	495,470.7*** (115,206.1) [269,659.6, 721,281.9] <.000>	155,229.7*** (34,180.5) [88,233.7, 222,225.6] <.000>
Mean of outcome	256,408.2	344,003.8	435,191.6	529,353.7	382,125.9
Package obs.	23,965	23,965	23,965	23,965	23,965
Day obs.	1	1	1	1	146
Package-day obs.	23,965	23,965	23,965	23,965	3,498,890

Note: Parentheses show standard errors clustered by packages, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. $^{*}p < .05$;$^{**}p < .01$;$^{***}p < .001$.

B.4: GitHub Events as Outcomes

In this section, we use the seven GitHub activity metrics (stars, forks, pushes, pull requests, opened issues, closed issues, and releases) as outcomes. Specifically, we estimate whether treated groups have a difference in those metrics at a post-treatment snapshot (July 1) and whether there is a linear trend across the entire post-treatment period (to the end of 2023). We find null post-treatment effects using the same post-treatment snapshot dates across all seven metrics (Table B.5 and Table B.6).

Table B.5: **PyPI Experiment: Post-treatment differences in GitHub repository event activity—stars, push events, and pull requests (ITT).** Odd-numbered columns report differences in means at a snapshot approximately three weeks after intervention ended (1 Jul 2023). Even-numbered columns report post-treatment differences in means over the full post-treatment period, allowing for heterogeneous treatment effects through a linear time trend.

	(1)	(2)	(3)	(4)	(5)	(6)
	Stars		Push events		Pull requests	
	Jul 1	Full	Jul 1	Full	Jul 1	Full
Treatment group	0.0	-1.0	7.7	6.7	9.2	8.0
	(7.7)	(7.4)	(7.5)	(6.7)	(5.6)	(4.9)
	[−15.1, 15.1]	[−15.4, 13.5]	[−6.9, 22.4]	[−6.3, 19.8]	[−1.8, 20.2]	[−1.6, 17.5]
	< .999 >	< .897 >	< .300 >	< .314 >	< .100 >	< .103 >
Linear trend		0.1***		0.3***		0.2***
		(0.0)		(0.0)		(0.0)
		[0.1, 0.1]		[0.2, 0.3]		[0.2, 0.2]
		< .000 >		< .000 >		< .000 >
Treatment group × Linear trend		0.0		0.1		0.1
		(0.0)		(0.0)		(0.0)
		[−0.0, 0.1]		[−0.0, 0.1]		[−0.0, 0.1]
		< .251 >		< .177 >		< .106 >
Constant	28.0***	25.7***	51.2***	44.9***	35.3***	30.8***
	(5.9)	(5.9)	(3.1)	(2.8)	(2.2)	(1.9)
	[16.3, 39.6]	[14.2, 37.3]	[45.1, 57.4]	[39.5, 50.4]	[31.0, 39.6]	[27.0, 34.6]
	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >
Outcome median/mean	0	0	0	0	0	0
Package obs.	23,916	23,916	23,916	23,916	23,916	23,916
Day obs.	1	23	1	23	1	23
Package-day obs.	17,872	3,681,632	17,872	3,681,632	17,872	3,681,632

Note: Parentheses show standard errors clustered by packages, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

Table B.6: **PyPI Experiment: Post-treatment differences in GitHub repository event activity—issues opened, issues closed, fork events, and release events (ITT).** Similar to Table B.5, but for issues opened/closed, fork events, and release events.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
	Issues opened		Issues closed		Fork events		Release events	
	Jul 1	Full	Jul 1	Full	Jul 1	Full	Jul 1	Full
Treatment group	1.3	1.0	1.2	1.0	0.6	0.4	0.3	0.2
	(1.4)	(1.2)	(1.1)	(1.0)	(2.0)	(1.9)	(0.3)	(0.3)
	[−1.4, 4.0]	[−1.4, 3.4]	[−1.0, 3.4]	[−0.9, 3.0]	[−3.3, 4.5]	[−3.4, 4.1]	[−0.3, 0.8]	[−0.3, 0.7]
	< .347 >	< .423 >	< .283 >	< .301 >	< .766 >	< .853 >	< .330 >	< .377 >
Linear trend		0.0***		0.0***		0.0***		0.0***
		(0.0)		(0.0)		(0.0)		(0.0)
		[0.0, 0.0]		[0.0, 0.0]		[0.0, 0.0]		[0.0, 0.0]
		< .000 >		< .000 >		< .000 >		< .000 >
Treatment group		0.0		0.0		0.0		0.0
× Linear trend		(0.0)		(0.0)		(0.0)		(0.0)
		[−0.0, 0.0]		[−0.0, 0.0]		[−0.0, 0.0]		[−0.0, 0.0]
		< .128 >		< .189 >		< .138 >		< .251 >
Constant	7.2***	6.4***	6.1***	5.4***	8.1***	7.5***	1.4***	1.3***
	(0.8)	(0.7)	(0.6)	(0.5)	(1.6)	(1.6)	(0.1)	(0.1)
	[5.7, 8.7]	[5.0, 7.8]	[4.9, 7.2]	[4.3, 6.5]	[5.0, 11.2]	[4.4, 10.5]	[1.2, 1.7]	[1.1, 1.5]
	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >	< .000 >
Outcome median/mean	0	0	0	0	0	0	0	0
Package obs.	23,916	23,916	23,916	23,916	23,916	23,916	23,916	23,916
Day obs.	1	23	1	23	1	23	1	23
Package-day obs.	17,872	3,681,632	17,872	3,681,632	17,872	3,681,632	17,872	3,681,632

Note: Parentheses show standard errors clustered by packages, square brackets show 95% Confidence Intervals (CI), and angled brackets show p-values. *$p < .05$;** $p < .01$;*** $p < .001$.

Appendix C: PyPI Observational Analysis

In this section, we present observational evidence using historical download data to understand the impact of human and bot downloads on human downloads. Using vector auto-regressive (VAR) models, we find that past human downloads predict future human downloads.

C.1: Sample and Data

We started by retrieving the full enumeration of all Python packages available from the PyPI repository. The index is extensive, with 458,274 packages at our retrieval time. We randomly sampled fifty thousand packages to keep querying costs and data processing tractable. We then queried daily downloads over a 3-month period (going back 91 days from our query date) from BigQuery and successfully retrieved N = 40,565.[16] We have the daily downloads split by package installer for each package. We classified installers into bot versus human downloads (Section 3.6). We filtered our dataset to packages with at least 50 human downloads over the three months, which gave us 13,481 packages and 1,226,771 package-day observations. So, our final sample is of Python packages that have been downloaded somewhat consistently over the last three months.

C.2: Research Design

To estimate how past downloads predict future downloads, we estimate a VAR model for each package to understand the non-experimental correlational evidence. Specifically, we want to understand the extent to which past downloads can explain future downloads.

The VAR model of each package has two equations: one where human downloads at time t is the outcome, and the other where bot downloads at time t is the outcome. Both outcomes are modeled as a function of their past values (e.g., human downloads) and the past values of the other (e.g., bot downloads) up to a select number of temporal lag ($t - p$). The maximum number of lags is three weeks (21 days) to allow downloads up to three weeks old to affect present downloads. The Akaike Information Criterion then determines the number of lags (p) for each package. We implement this using *statsmodels* (Seabold and Perktold 2010). Some packages do not have a fitted model because of numerical instability or unit roots. For packages that successfully converge with an optimal lag of at least a day (n = 6630), we further do a Granger causality test where the null is that past values of a time series do not collectively predict future values. From this, we get n = 6620 p-values from the Granger test (10 models fail to compute from numerical issues, Seabold and Perktold (2010)). Having a p-value that rejects the null at conventional levels does not mean that past downloads cause future downloads.

16. https://warehouse.pypa.io/api-reference/bigquery-datasets.html.

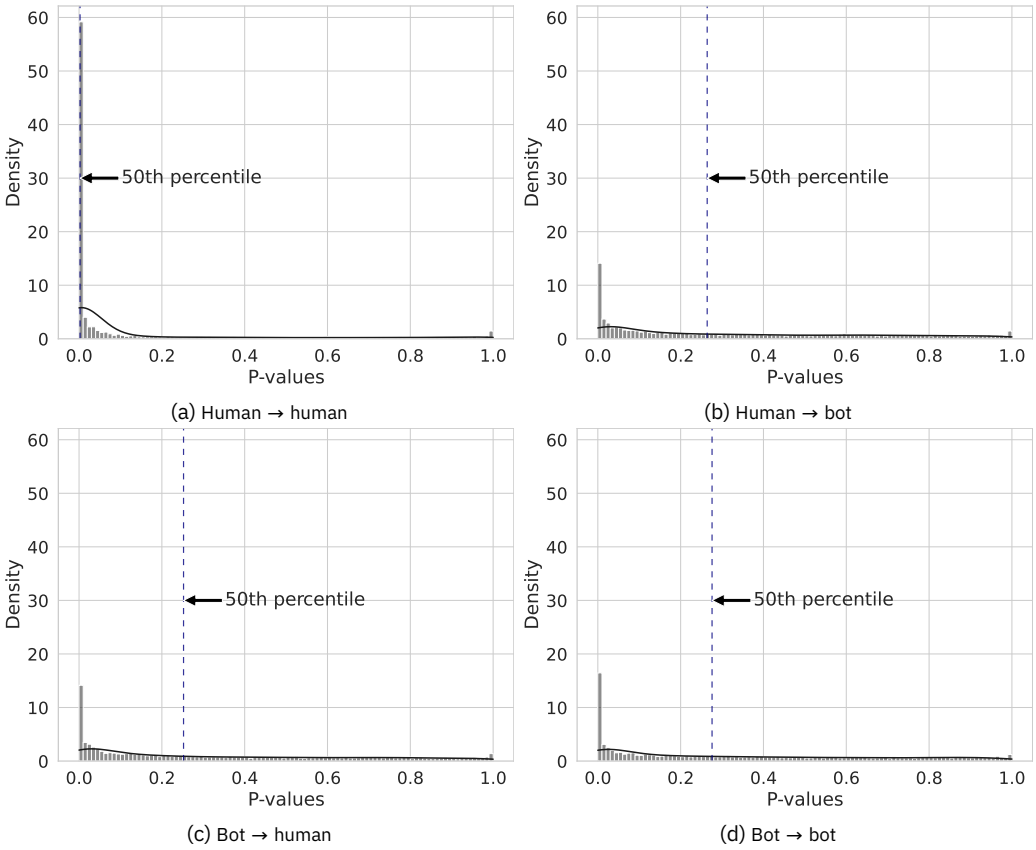

Figure C.1: **Distribution of P-values From Granger Causality Tests.** The figure plots the distribution of p-values for each of the four versions (by panels) of the Granger tests for 6,620 packages and 602,420 package days. All plots have the same scale. Each package's downloads over three months are first estimated in a VAR model, with human downloads and bot downloads as the two outcomes. Estimated coefficients are then used to implement the Granger causality tests. P-values are binned into 100 equal-width bins, each with a range of .01. The vertical dashed line indicates the 50th percentile value in the p-values.

C.3: Results

Figure C.1 reports the collection of p-values from all the successful VAR runs for the four versions of the Granger tests. The vertical dashed line indicates the 50th percentile in p-values. Overall, we observe that only past human downloads can predict human downloads (Figure C.1a). Its 50th percentile in p-values is approximately .002, implying more than 50% of the packages' VAR model have p-values small enough to reject the null hypothesis at the 1 percent level that past human downloads do not predict human downloads. Specifically, the percentile value for a p-value of .05 is the 69th percentile, suggesting that human downloads are associated with yet more human downloads.

This same observation is not true for the three other versions (Figures C.1b to C.1d). Some packages have Granger tests where the p-values are small enough to reject the null at conventional levels. However, they are in the minority, with the 50th percentile

p-values that are much higher. For these three groups, a p-value of .05 falls at only the 25th, 26th, and 26th percentiles, respectively.

In all, we conclude that past human downloads predict future human downloads. However, past human downloads do not predict bot downloads, and past bot downloads predict neither human nor bot downloads.