Managing by Feedback

Jinci Liu *

October 27, 2025

(Job Market Paper - Latest Version Available Here)

Abstract

This paper studies how managers affect workers through feedback. I analyze feedback during software code reviews and measure developers' productivity and retention using GitHub and LinkedIn data. The dataset covers over 200 million feedback messages from 1.7 million teams. Using text classification methods, I classify feedback into different types along tone and information dimensions (e.g., toxic, positive, and constructive) and estimate their causal effects on developers' outcomes. My instrumental variables design exploits the fact that some teams adopt random code review assignments. I find that toxic feedback reduces developers' future code quantity and quality, whereas respectful criticism (negative but non-toxic) has no such detrimental effects. Positive feedback increases future code quality, raises retention, and generates spillovers to coworkers and other teams. Constructive feedback reduces future code quality because developers focus more on revising old code, which crowds out new code development. Finally, I measure manager quality by their contribution to developer productivity growth and show that feedback explains a substantial share of the variation in manager quality.

*IIES, Stockholm University. Email: jinci.liu@iies.su.se.

I am grateful to Arash Nekoei, Mitch Downey, and Jósef Sigurdsson for their encouragement and feedback—always positive and never toxic. I also thank Huen Tat Au-Yeung, Kirill Borusyak, Konrad Burchardi, Stefano DellaVigna, Ingrid Haegele, Martin Koenen, Patrizia Massner, Chloé Nibourel, Torsten Persson, Nancy Qian, David Schönholzer, Mateusz Stalinski, David Strömberg, Horng Chern Wong, and Yimei Zou, as well as seminar participants at IIES, SOFI, SSE, the HK Labor Symposium, and CEPR-IMO & ESF. I owe special thanks to Jakob Beuschlein for his insightful feedback on instrumental variable designs. I also thank my developer friends for their patience, even when their constructive feedback made me redo many things. I thank Ruipu Peng and Peishan Ju for excellent research assistance in validating the LLM classifications. I acknowledge support from Google Cloud Research, and I am grateful to the Mannerfelt Foundation and the Institute for Evaluation of Labour Market and Education Policy (IFAU) for their generous financial support. All errors are my own.

1 Introduction

How can we increase worker productivity? Economists have studied several drivers, such as technology, human capital, and incentive contracts (Solow, 1957; Becker, 1966; Holmström, 1979). A growing literature shows that managers play a crucial role in affecting worker productivity, but less is known about how they achieve this effect.[1] This paper focuses on feedback, a central practice through which managers interact with workers.

Feedback is ubiquitous in organizations. Over 90% of managerial jobs involve giving feedback (O*NET, 2025) and McKinsey even calls feedback "the fuel of development" (McKinsey, 2024). However, its effectiveness is unclear. In one survey, 80% of workers felt more engaged after receiving "meaningful" feedback (Gallup, 2022). In another, only 14% strongly agreed that the feedback they received is helpful (Sutton and Wigert, 2019). One reason for this mixed evidence is that the impact of feedback depends on how workers interpret it and adjust their effort. For example, toxic feedback can damage confidence but some theories suggest that workers may also increase effort to avoid further punishment (Mirrlees, 1974, 1976). Ex-ante, the effect of feedback on productivity is ambiguous.

This paper studies how managers affect worker through feedback, and how feedback contributes to manager quality. In the first part, I use large language models to classify feedback by its tone and informational content, and provide causal evidence that different types of feedback have distinct effects on worker productivity and retention. In the second part, I measure manager quality as value-added (VA) to worker productivity and use machine learning to show that feedback quantity and quality together explain 22% of the variation in manager VA.

Studying feedback presents four major challenges. First, most feedback is verbal and rarely recorded. Second, even when written, feedback has multiple dimensions, making it conceptually difficult to decide which to study and technically demanding to classify them at scale. Third, identifying the causal effects of feedback on productivity is difficult. Workers who receive different feedback may differ in unobserved ability, so the correlation between feedback and productivity may be due to selection or reverse causality. Finally, measuring manager quality is challenging, since firms may assign more productive workers to particular managers.

I use written feedback from code reviews in software development, which play an important role in modern economic growth.[2] In this setting, *developers* submit code for integration into the team's codebase, and *reviewers*, typically senior team members, act as middle managers. They provide feedback and decide whether to approve or request revisions. Developers must address reviewer feedback before code integration. I use data from global software teams on GitHub, the world's largest coding platform. Over 90% of Fortune 100 companies use it (GitHub, 2025).

[1] See, for example, Ichniowski et al. (1997); Bloom and Van Reenen (2007); Weidmann et al. (2025).

[2] U.S. Bureau of Economic Analysis (2025) estimates that the digital economy accounts for about 10% of US GDP and 8.9 million jobs in 2022.

My analysis focuses on professional developers at firms such as Google and Microsoft. Using GitHub Archive, I measure developer productivity as weekly code quantity (lines of code) and quality (share accepted). I then link GitHub to LinkedIn for long-term outcomes. The dataset covers 1.7 million teams and 231 million feedback messages from 2017 to 2023.

I classify feedback along tone and information using large language models (BERT-based models and GPT). I focus on three interpretable dimensions: toxicity, positivity, and constructiveness. These dimensions are not mutually exclusive, and I also examine their interactions.[3] The toxicity and positivity dimensions capture tone, while the constructiveness dimension reflects information. Toxic feedback is intentional harm, positive feedback provides encouragement or acknowledgment, and constructive feedback offers specific, actionable information.

To identify the causal effects of feedback, I exploit the random assignment of code cases to reviewers and variation in reviewer feedback style. The strategy is analogous to judge randomization, where treatment assignments vary because some judges are systematically harsher or more lenient (Dobbie and Song, 2015; Bhuller et al., 2020; Agan et al., 2023). I construct a measure of reviewer style for each feedback type, defined as the reviewer's average tendency to give that type of feedback in other developers' reviews. I then use this measure as an instrument for the feedback a developer receives. Some teams use scripts that randomly allocate reviewers to reduce personal bias and balance workloads, creating natural variation in the feedback workers receive. I identify these teams by detecting such scripts in their codebases.

My main finding is that feedback has significant effects on workers' future productivity and retention.[4] Toxic feedback reduces future productivity substantially. In the month after a toxic review, developers produce 42.9% less new code for their teams. Developers do not reallocate to non-code tasks or move to other teams to avoid toxic reviewers. Instead, the overall output quantity decline. Additionally, there is no evidence of a quality–quantity tradeoff. The decline in code quantity is not offset by higher code quality. In fact, the quality of their new code also falls by 13.1%. Because most toxic feedback is also negative, a key question is whether the decline stems from toxicity or negativity. The distinction matters for policy: if all criticism is harmful, managers would need to avoid any negative feedback. I find that the harm of toxic feedback does not come from expressing dissatisfaction with the work. After respectful criticism (negative but non-toxic), developers increase both coding and non-code quantity.

Positive feedback, by contrast, increases new code quality by 6.4%. Its benefits extend beyond the affected team. Developers who receive positive feedback are more likely to send positive messages to coworkers and produce higher-quality code for other teams in the following

[3]Each feedback message is coded along three binary variables corresponding to these dimensions: toxic or non-toxic, positive or negative, and constructive or non-constructive. A single message can be toxic, negative, and constructive. See Section 3 for details.

[4]To avoid contamination from the review itself, I exclude the current code being reviewed and focus only on new code produced afterwards.

month. Thus, positive feedback increases productivity directly and through organizational spillovers. These effects are consistent with behavioral models where higher self-confidence increases effort (Bénabou and Tirole, 2002; Compte and Postlewaite, 2004; Kőszegi et al., 2022). In this setting, positive feedback may increase productivity by strengthening self-confidence. Developers who receive positive feedback are also more likely to remain in the tech industry over the next one to two years. Since worker knowledge is a key asset, and turnover represents a major loss (Hoffman and Tadelis, 2021), this finding suggests that positive feedback can help improve retention in knowledge-intensive firms.

Regarding the informational dimension of feedback, constructive feedback has more nuanced effects. It does not change the quantity of new code but reduces new code quality. One explanation is that reviewers not only provide feedback, but also decide whether the code is accepted. When developers receive detailed or demanding feedback, they must rework old code. This crowds out time for writing new code and can compromise its quality. Consistent with this mechanism, developers rewrite 49.2% more lines of old code after receiving constructive feedback, suggesting a trade-off between revising old code and developing new code.

The interaction between the tone and information of feedback plays a critical role in shaping developer outcomes. Most benefits of positive feedback comes from positive but not constructive feedback. This implies that motivation from the tone, rather than specific information, drives the effect. The effects also differ across developer groups. Toxic feedback reduces productivity for most developers but increases new code quantity among Asian developers, though new code quality declines for all. Men respond more negatively than women. However, women represent only 12% of the sample, so these differences are suggestive.

In the second part of the paper, I analyze how much feedback explains variation in manager quality. I begin by measuring manager quality as value-added (VA) to worker productivity. To estimate VA, I apply the forecast-based estimator of Chetty, Friedman, and Rockoff (2014), following the teacher literature. Reviewer VA is defined analogously as the change in developer productivity before and after assignment, measured by both code quantity and quality.

I first estimate reviewer quality in the random assignment sample and find positive correlation between two measures: changes in developer code quantity and code quality. I then extend the analysis to the full sample. To test validity, I focus on reviewers who work across multiple teams and appear in both random and nonrandom samples. Their VA rankings closely align, supporting that the forecast-based estimator reduces bias and work beyond random assignment.

After confirming the VA measure's validity, I test how much manager VA variation feedback explains. Managers affect workers through multiple channels, including monitoring performance, reallocating tasks, and motivating effort (Weidmann et al., 2025). VA captures managers' overall effects not only feedback. To test the explanatory power of feedback, I train models using three sets of covariates: feedback quantity, feedback quality, and selected features

of feedback. Feedback quantity is measured by the number of feedback messages a reviewer sends per week, while feedback quality is summarized using text-based features that capture the linguistic feedback. A random forest model shows that feedback quantity and quality together explain about 22% of the variation in reviewer VA. In contrast, reviewer gender and race explain only 0.2%. Remarkably, a simple model using only three feedback types—toxic, positive, and constructive—captures 84% of the explanatory power of the full text-based features.

The paper shows that feedback does more than transfer information. Managers can use feedback to affect worker productivity. Feedback explains a substantial share of manager quality.

Related literature. This paper contributes to several strands of literature. First, this paper contributes to the literature on feedback. Feedback has been widely studied in education and psychology. A widely used definition conceptualizes feedback as information provided by agents (e.g., teachers, peers, parents) regarding aspects of one's performance or understanding. In education, teacher feedback is central to student achievement (Hattie and Timperley, 2007; Banihashem et al., 2022). In psychology, feedback is linked to feedback-seeking behavior as a driver of learning (Ashford and Cummings, 1983; Kluger and DeNisi, 1996; London, 2003).Recently, economics has begun to study feedback as a channel for human capital accumulation (Emanuel et al., 2025). A related literature documents gender differences in how feedback is given and received. Women tend to interpret negative feedback as a signal of low ability, while men attribute it to luck (Shastry et al., 2020). Such asymmetries contribute to gender gaps in confidence and participation. Other studies show that women evaluators face stronger resistance to negative feedback (Saygin et al., 2023), and that reducing public signals of gender-biased feedback increases women's output and visibility (Freimane, 2024). I extend this literature by providing large-scale causal evidence on how feedback affects worker productivity, generates spillovers, and varies across workers with different characteristics and experiences.

Second, this paper contributes to the literature on incentives and worker performance. Organizations traditionally rely on monetary incentives such as wages, bonuses, and promotions to reward workers' effort and punish their shirking. Theory predicts that rewards raise the marginal benefit of effort, while punishments deter shirking (e.g., Holmström (1979); Holmstrom (1982); Shapiro and Stiglitz (1984)). Consistent with these predictions, empirical studies document the effects of performance pay and promotion incentives on worker performance (Lazear, 2000; Gibbons and Waldman, 1999), and experiments further show that both rewards and punishments can sustain cooperation, with punishments more effective (Andreoni et al., 2003). However, monetary incentives are costly, constrained by budgets, and limited by the number of available promotions. Furthermore, pay inequality can be detrimental to morale, productivity, and employee retention (Card et al., 2012; Dube et al., 2019). Feedback has none of these restrictions and it can be delivered freely and frequently. Positive feedback acts

like a reward and toxic feedback acts like a punishment. Thus, feedback serves as a distinct non-monetary incentive mechanism and this paper provides evidence that such non-monetary incentives can affect worker outcomes.

Third, the paper relates to behavioral economics, which shows that emotions and self-confidence influence effort (Bénabou and Tirole, 2002; Compte and Postlewaite, 2004; Kőszegi et al., 2022), as well as to models of communication, which demonstrate that messages can shape behavior(Crawford and Sobel, 1982). I provide empirical evidence consistent with these theories by showing that the tone of feedback can worker productivity.

This paper also contributes to research on how individual managers affect worker outcomes (e.g., Ichniowski et al. (1997); Bertrand and Schoar (2003); Bloom and Van Reenen (2007); Bandiera et al. (2007); Lazear et al. (2015); Bandiera et al. (2020); Frederiksen et al. (2020); Adhvaryu et al. (2023); Metcalfe et al. (2023); Weidmann et al. (2025)). While existing studies highlight channels such as people management skills (Hoffman and Tadelis, 2021), talent identification (Haegele, 2022), and task allocation (Minni, 2024), I add to this literature by uncovering feedback as an important but understudied channel in which managers affect worker productivity.

Finally, this paper contributes to research on non-wage job amenities and toxic workplace conditions. Economic studies have examined toxic speech outside the workplace, such as on social media and online forums (Smirnov et al., 2023; Ederer et al., 2024), and adverse conditions within firms, including workplace hostility (Collis and Van Effenterre, 2025), sexual harassment (Folke and Rickne, 2022), and violence (Adams-Prassl et al., 2024). These studies often emphasize gender asymmetries. Related work outside economics, largely based on surveys and qualitative evidence, shows that toxic environments link to depression, burnout, and absenteeism (Nielsen and Einarsen, 2012; McTernan et al., 2013; Fattori et al., 2015; Zeltzer et al., 2023). I extend this literature by providing causal evidence on the negative effects of toxic feedback in the workplace.

The rest of the paper is organized as follows. Section 2 describes the data sources and the context of software development. Section 3 explains the feedback classification. Section 4 presents descriptive evidence on team structure and reviewer characteristics. Section 5 outlines the empirical design, and Section 6 presents the main results on how feedback affects productivity. Section 7 examines reviewer value-added, and Section 8 concludes.

2 Data and Setting

I create a panel dataset by merging three sources: GH Archive (public timelines, activities, feedback, and team structure), GitHub profiles (names, images, locations), and Revelio Labs (employment histories). I use the data to (i) characterize reviewer and developer demographics;

(ii) identify reviewer-to-developer feedback messages; and (iii) measure feedback types, developer productivity, and retention. The panel spans 2017 to 2023 and includes teams that adopt random reviewer assignment.

2.1 GitHub: World's Largest Software Coding Platform

GitHub is the largest online coding platform, where developers store code, share projects, and collaborate. As of June 2025, it hosts over 259 million public repositories and more than 161 million users.[5] Its estimated global value exceeds 8 trillion USD (Hoffmann et al., 2024). Technology firms such as Google and Microsoft maintain thousands of public projects on Github. These firms increasingly rely on GitHub for innovation and productivity (Nagle, 2019).

GitHub records the full version history of each file and supports asynchronous collaboration across time zones and teams. The GH Archive captures all public GitHub activity in near real-time and preserves event-level traces, even after code is modified or deleted.

To join GitHub, each usermust create a unique username and can publicly display personal information on the GitHub profile. The GitHub profile usually lists name, profile image, location, firm, bio, and email. Many highlight programming skills to attract potential employers, often providing accurate details and links to LinkedIn profiles or personal websites to signal credibility and professional identity (El-Komboz and Goldbeck, 2024).

Studying workplaces on GitHub is challenging because millions of volunteers contribute, and many activities are sporadic. To focus on professional settings, I include only teams owned by organizations where developers work in teams with defined roles. I exclude individual projects outside organizational teams to ensure the sample reflects structured workplace collaboration, not ad hoc or personal coding. Below, I define "team," "team member," and "reviewer," and then describe the final sample restrictions used in the analysis.

2.1.1 Team

A repository is a workspace where developers store code and track its version history. I define a *team* as a repository owned by an organization. In these teams, developers contribute code, review, and discuss.[6] I restrict the analysis to organization-affiliated teams for three reasons. First, they use structured workflows. Second, they align with firm objectives such as

[5]See user and repo, accessed 2024-06-29.

[6]The concept of a "team" has been defined in various ways in economics. Marschak and Radner (1958) defines a team as "an organization the members of which have only common interests," highlighting the alignment of incentives. Holmstrom (1982) defines a team "rather loosely as a group of individuals who are organized so that their productive inputs are related," emphasizing joint production. In more recent work, such as Jones (2021), teams are observed through patterns of co-authorship in science. Following this definition, I define each GitHub repository as a team, a group of developers jointly contributing code toward a shared project, much like a group of researchers co-authoring a paper.

product development or commercialization (Andersen-Gott et al., 2012). Third, they better represent workplace settings than individual projects, which often reflect personal work. Figure 1a illustrates the distinction between organization-affiliated teams included in the sample and individual projects excluded from the sample.

[Figure 1 about here.]

2.1.2 Team Member

Teams include internal members and outside volunteers. Anyone can comment or submit code changes, but only internal members hold project-level permissions. I identify internal members using GitHub's *author association* field, available since 2017, which classifies each user as *owner*, *member*, *collaborator*, *contributor*, *none*, or *mannequin*.

I define a team member as a user labeled *owner*, *member*, or *collaborator*. These roles imply write-level access and active involvement in development, including pushing code, reviewing pull requests, managing issues, and coordinating work.[7]

Because I only observe *author association* when a user acts, months where no activity occurs are missing, making it difficult to directly track periods of continuous membership. To address this, I impute continuous membership between the first and last months in which a user is observed as a team member. This approach is justified by the rarity of exits and re-entries within the observed window.

Demographics. I measure developers' gender, race, and location using information from their GitHub profiles. Gender is inferred using a deep learning model on profile images, with predictions below 0.85 confidence being coded as missing. Around 88% of developers are classified as male. Race is classified from developers' names using standard name-based methods into four categories: Asian, White, Black, and Missing. Location is taken from self-reported profile entries and standardized at the country level. Figure A1 maps the geographic distribution of team members. The United States has the largest concentration, followed by India, China, and Brazil, with notable participation also seen in Europe and Southeast Asia. Further details are provided in Appendix A.

2.1.3 Case, Reviewer, and Feedback

A pull request on GitHub is a formal submission of code changes for integration into a team's codebase. I refer to each pull request as a *case*. Teams often use protected branches that require

[7] *Owner* and *member* are formally affiliated with the organization. *Collaborator* is an invited contributor with similar technical access. Users labeled *contributor* or *none* are external or occasional participants without sustained permissions. *Mannequin* is a placeholder account and is excluded. This measure is preferred to GitHub's `MemberEvent`, which captures only a subset of membership changes.

code reviews before merging. Submitting a case initiates a structured review process. Code review is a standard practice in professional teams. It helps identify bugs, enforce conventions, and maintain quality. [8]

Reviewers as team members formally listed in the "reviewer" field of the case. They are responsible for evaluating the code and providing feedback. Although a case may have multiple reviewers, 80.5% have only one. *Developers* as the team members who submitted the case or contributed code on the same day the feedback was posted. 46.9% of cases have one developer. These definitions reflect the collaborative nature of software development. Like coauthors responding to referee reports, developers jointly engage with reviewer feedback to improve the submission.

Reviewer messages are collected from GH Archive, which records all public GitHub events in real time. Even if a user deletes a message from the GitHub interface, GH Archive preserves the original text. Figure 1b presents a schematic of feedback exchange within a case. Figure 1c illustrates an example from the data.

I do not observe formal managerial relationships in the data. Many reviewers also write code themselves. However, Section 2.5 shows that reviewers are typically senior.[9] I further document hierarchical feedback patterns within teams. Most teams follow a hierarchical structure: reviewers provide feedback to others but rarely receive feedback from those developers. These patterns suggest that reviewers act as de facto middle managers or project leads, coordinating work and ensuring code quality. Moreover, my results are nearly identical if I restrict to only very hierarchically structured teams.

2.2 Reviewer Assignment

When a developer submits a case that requires review, GitHub recommends reviewers based on prior contributions to the affected code. These recommendations rely on *git blame*, which tracks the last person to modify each line in the file. Reviewers with recent contributions are more likely to understand the context and catch errors. The developer can select a reviewer by clicking "Request" next to a suggested name. They can also type in a team member's username.

2.2.1 Random Reviewer Assignment

In teams that handle many cases, review workloads often become uneven, which can reduce efficiency and slow collaboration. To address this problem, some teams adopt random

[8] In the sample, 17.11% of cases have reviewers. Emanuel et al. (2025) discuss how reviews raise quality and facilitate learning; job postings at Microsoft and Google explicitly require code review.

[9] Based on the matched GitHub–LinkedIn sample, reviewers often hold titles such as "Principal Software Engineer" or "Senior Software Engineer".

reviewer assignment tools. These tools balance workloads by rotating reviewers based on recent activity.[10] As one developer explained: "Many development teams struggle with uneven review distribution. A tool that assigns pull-request reviews according to each member's recent activity could help balance the load" (see Figure B1).

GitHub does not provide random reviewer assignment by default. However, teams can implement it using GitHub Actions and configuration files such as `reviewer-lottery.yml`, `assign-reviewers.yml`, or `CODEOWNERS`. These files specify eligible reviewers and rotation rules. When a case is opened, the Action reads the configuration and assigns a reviewer according to past workloads. Table B1 summarizes the functions of these files, and Figure B2 illustrates two scenarios: one with the random assignment and one with manual selection through "git blame".

Under random reviewer assignment, the match between submitters and reviewers is plausibly exogenous. Allocation depends solely on recent workloads rather than case content or submitter identity. In Consistent with this, the persistence of reviewer–developer matches is markedly low. Conditional on a developer having been reviewed by a given reviewer, the probability of being reviewed by the same reviewer in the following case is 7.32%. Section B details the assignment rules and compares teams that adopt this system with those that do not. In Section 5.1, I show that this assignment is uncorrelated with case and developer characteristics.

2.3 LinkedIn Profiles Data

To measure career outcomes, I link developers to employment histories from Revelio Labs. Revelio compiles information from LinkedIn and other public sources, producing a panel of 1.25 billion professional profiles worldwide. The data record job histories with start and end dates for each firm–position spell as of August 2024.[11] Appendix E details the GitHub–LinkedIn matching procedure.

I use these data to construct measures of career transitions. I define a *firm switch* as a move to a new employer, regardless of industry. I define an *industry exit* as a transition to a firm in a different two-digit NAICS sector.

2.4 Developer Productivity Measures

I measure developer productivity along two dimensions: code quantity and code quality. Code quantity is the number of lines added to the team's codebase. Code quality is measured using the code acceptance rate and case correctness rate. I also track non-code output to capture potential task reallocation in response to different types of feedback.

[10] See discussion, accessed 2025-01-31.

[11] A random sample of 1,000 developers shows that 71.2% have LinkedIn profiles, and 97% of these are also in the Revelio Labs.

Code Quantity. The main output variable for code quantity is the **lines of code** a developer submits to a team each week. I also calculate the number of rewritten lines to capture revisions to old code. Details are in Appendix A.4. While more lines of code do not necessarily indicate higher productivity, line counts remain a standard proxy for productivity in the literature (e.g., Vasilescu et al., 2015; Wagner and Ruhe, 2018; Emanuel et al., 2025).

Code Quality. I measure output quality using two indicators. The first is the **code acceptance rate**, defined as the fraction of initially submitted lines that are accepted and retained in the team codebase. A higher value implies that the original code required fewer edits and reflects greater accuracy of the first submission (McIntosh et al., 2016; Emanuel et al., 2025). The second is the **case correctness rate**, defined as the share of cases whose initial submission is merged into the team's main codebase. A higher rate indicates that developers' work meets review standards without major revisions. Appendix A.5 provides definitions and examples. Figure F4 shows a positive correlation between the' number of developers ' followers on GitHub and both quality measures. The number of followers on GitHub is a signal of influence and professional reputation. This positive association supports the validity of the quality metrics.

Non-Code Quantity. I measure a developer's non-code quantity every week as the number of non-code GitHub activities, such as participating in issue threads, answering questions, or providing troubleshooting support.

2.5 Sample Restriction and Summary Statistics

After defining teams, members, and reviewers, I apply restrictions to ensure the sample captures structured workplace collaboration, not ad hoc activity. First, I restrict attention to teams under organizations. I exclude projects operated by individual users, as organizational teams are more likely to reflect workplace-like environments. Second, I retain only code submissions made by formal team members. Such members contribute more regularly and know the project well, making productivity effects more meaningful. Third, I focus on cases with at least one reviewer.

I construct two samples. The *full sample* includes all teams that satisfy these restrictions between 2017 and 2023. It covers 1,774,184 teams and 56,637,966 cases, which together generated more than 231,293,881 feedback messages.

The *random reviewer sample* includes 3,457 teams. Although this is a small fraction of the 1.7 million teams in total, these teams are highly active, accounting for 7% of all cases (4,228,687) and 11% of all feedback messages (24,453,479). They also tend to belong to leading technology firms. Table B2 reports the number and share of teams in major firms that adopt

GitHub's random reviewer feature.

Teams using random reviewer assignment are, on average, 20 times larger than those in the full sample. Their members are less likely to be in the same geographic area. Most feedback occurs online rather than in person. This sample, therefore, captures a particularly relevant subset of teams that drive innovation and growth. It also provides higher internal validity, since offline feedback channels are less likely to operate at the same time.

Comparing Random Reviewer Adopting Teams with Non-Adopters. Table 1 shows that teams adopting random reviewer assignment are systematically larger and handle far greater review volumes. On average, they generate 7073.61 feedback messages, over 50 times as many as other teams. They have nearly 20 members and process 1223.22 cases, over 50 times the number in other teams. These patterns suggest that adoption is concentrated in large teams with substantial review workloads, where coordination challenges are greatest, and random assignment helps distribute reviews more evenly.

[Table 1 about here.]

Comparing Reviewers and Developers. Within the random reviewer sample, I compare reviewers and developers along five dimensions: gender, race, platform reputation, experience, and activity. Figure F3 shows that reviewers are more likely to be male (89% vs. 86%) and less likely to be Asian (16% vs. 21%). They have greater platform visibility, averaging 133 followers compared with 83 for developers. They are also more experienced and active, recording an average of 1,844 yearly activities compared with 668 for developers. Overall, reviewers tend to be drawn from more experienced and prominent contributors. LinkedIn job titles for matched GitHub–LinkedIn profiles confirm this pattern: reviewers cluster in senior roles with *Senior Software Engineer* being the most common, followed by *Staff* and *Principal* positions. Developers, in contrast, are concentrated in more junior roles such as *Software Engineer* and *Intern*.

I then turn to developers' outcomes, summarized in Table 2. Productivity is measured along three dimensions: code output, code quality, and non-code activities. Because output distributions are heavy-tailed, with bursts of submissions or large rewrites after major changes, I winsorize all continuous variables at the 95th percentile while keeping bounded outcomes such as case correctness and code acceptance rates unchanged.

Table 2 further distinguishes between the "Focal Team," defined as the team where the developer received reviewer feedback on a specific case in a given week, and "Other Teams," which aggregates the developer's activity across all other teams in the same week.

[Table 2 about here.]

3 Feedback Classification

This section analyzes reviewer feedback using large language models (BERT-based models and GPT). Manual classification of text is difficult to scale because it is labor intensive and inconsistent across raters, making it unreliable for studying millions of feedback messages. I therefore use a model-based approach to classify feedback systematically based on linguistic features and validate the results with human annotation.

Feedback is multidimensional and can be interpreted from various perspectives, particularly with the flexibility offered by LLMs. Before focusing on the tone and information dimensions, I first provide a broad overview of feedback patterns in the data. I apply principal component analysis (PCA) to the complete corpus of reviewer feedback. Unlike supervised approaches that depend on predefined labels, PCA uncovers the main axes of variation directly from the text. The analysis reveals two dominant dimensions: positivity (for example, good, thank) and constructiveness (for example, add, test), as shown in Figure C1.

3.1 Pre-processing

To ensure reliable measurement, I first clean and standardize the text by removing quoted blocks from other users' messages to prevent misattribution, as well as code snippets. Next, I replace commonly used acronyms in software engineering, such as *"lgtm" (look good to me)* and *"kiss" (keep it simple)*, with their full forms. I then account for software-specific terminology that might be misclassified in general English, noting that words like "kill," "trash," and "dump" may refer to legitimate coding operations but are often flagged as toxic in general-purpose models (Sarker et al., 2020).

3.2 Toxic Feedback

The first dimension is toxicity. It captures whether feedback is intentional harm. The Cambridge Dictionary defines "toxic" as "very unpleasant or unacceptable, causing you a lot of harm and unhappiness over a long period." Toxicity within reviewer feedback is classified using the ToxiGen-RoBERTa model (Hartvigsen et al., 2022). Compared to keyword-based classifiers, ToxiGen is capable of detecting both explicit and implicit forms of toxicity. In this model, toxicity refers to the intentional harm (offensive or rude humor, insult, personal attacks, profanity, aggression), sexual harassment, and discrimination. This definition follows the guidelines used in human annotation, as illustrated in Figure C2, which shows the evaluation form used in the ToxiGen dataset. The model, a fine-tuned checkpoint of RoBERTa (Liu et al., 2019), achieves 94.5% accuracy and has been applied in economics (Ederer et al., 2024) and in

workplace contexts, for example, by Microsoft to classify harmful communication.[12]

ToxiGen specifically distinguishes between blunt critique and personal attack. For example, *"This is bad code"* is classified as non-toxic (probability: 0.918). When a suggestion is added, as in *"This is bad code. You can improve it by adding a loop"*, the confidence rises to 0.988. Similarly, *"Your paper is not of general interest to publish at QJE"* is labeled non-toxic (0.999).

In contrast, feedback targeting the individual is flagged as toxic. For example, *"what's wrong with you to write such bad code?"* is toxic (0.712). Implicit attacks, such as *"We can learn nothing from your code"* (0.674) or *"You'll never survive the job market with work like this"* (0.986), are also classified as toxic. Finally, discriminatory content such as *"Women are just not good at coding"* is flagged with high confidence (0.988). Figure F23 shows examples of toxic feedback and replies. Below are selected examples with ToxiGen-predicted toxicity scores:

1. *This is bad code* → non-toxic, probability: 0.918
2. *Your paper is not of general interest to publish at QJE* → non-toxic, probability: 0.999
3. *What's wrong with you to write such bad code?* → toxic, probability: 0.712
4. *Women are just not good at coding* → toxic, probability: 0.988
5. *Reject. We can learn nothing from your paper* → toxic, probability: 0.674
6. *You'll never survive the job market with work like this* → toxic, probability: 0.986

3.3 Positive Feedback

The second dimension is positivity. It captures whether feedback conveys a supportive or discouraging tone. I measure it using SiEBERT, a sentiment model trained on 12 million labeled documents (Hartmann et al., 2023). Compared with dictionary-based methods, SiEBERT is substantially more accurate across diverse domains and has used in economics (Brynjolfsson et al., 2025). The model classifies "This is bad code" as negative (0.999), and "Look good to me" as positive (0.996).

3.4 Constructive Feedback

The third dimension is constructiveness. It captures whether feedback provides specific and actionable information. The Cambridge Dictionary defines "constructive" as "advice, criticism, or actions that are useful and intended to help or improve something." Applying this definition in practice is difficult because judgments vary. For example, *"you can do it in a better way"*

[12]See https://www.microsoft.com/en-us/research/publication/toxigen, accessed 2025-10-14.

may seem constructive to some but vague or unhelpful to others. Prior work, such as Kolhatkar et al. (2020), labels constructive comments as "high-quality comments that contribute to the conversation." But this evidence comes from news discussions rather than technical platforms like GitHub, where feedback is shorter and highly specialized.

Detecting constructiveness requires reasoning beyond surface features of input text. Embedding-based models such as BERT[13] work well for toxicity and sentiment, where tone and word choice are decisive. Constructiveness, by contrast, depends on intent, technical content, and how suggestions affect the underlying code. For instance, judging whether *"consider adding a loop"* is constructive requires understanding the code's implementation. Input embeddings cannot capture this reasoning. GPT, trained on large and diverse corpora, can integrate broader context and domain knowledge. This makes them well-suited to this task.

I briefly outline the procedure here. Details are in Appendix C.2. Each feedback message is paired with the corresponding code submission. The model is prompted to assess whether the feedback provides specific and actionable information with a score from 0 (least constructive) to 10 (most constructive). To refine prompts and select the best model, I compare four OpenAI LLMs (gpt-4.1-nano, gpt-4o-mini, gpt-4o, and gpt-o3) against human labels. These labels are produced by a computer science master's student and verified by a software engineer. I then apply the best-performing model to the random reviewer sample. Using a threshold of 4 to define constructive feedback yields 90% classification accuracy. Table C2 provides examples.

The following examples illustrate the scoring:

1. *Agree* → Non-constructive, score: 0
2. *This is not clear* → Non-constructive, score: 1
3. *I don't think all introspector errors are reported in the domain status. We need to make sure that all severe jrf ones are* → Constructive, score: 4

3.5 Classification Results

Table 1, Panel C, compares classification results across the Full Sample and the Random Reviewer Sample.[14] The two samples yield similar patterns: about 0.21% of feedback is toxic, roughly 40% is positive, and less than 35% is constructive. Figure 2 illustrates these shares for the Random Reviewer Sample.

[Figure 2 about here.]

[13] An embedding is a numerical representation of text in a high-dimensional vector space, where semantically similar words or phrases are located close to each other.

[14] In the Random Reviewer Sample, all feedback messages are classified. In the Full Sample, toxicity and positivity are measured on all messages. Constructiveness is evaluated on a subsample due to budget constraints. Unless otherwise noted, classification results refer to the Random Reviewer Sample.

To show how these labels are assigned, Figure F22 plots the distributions of predicted scores. Panel (a) presents toxicity probabilities, which cluster between 0.8 and 1.0. This suggests that ToxiGen flags toxic feedback with high probability. Panel (c) plots the LLM's constructiveness scores. Manual validation confirms the measure's accuracy. Research assistants labeled a subsample and achieved 90% agreement with the Panel (d) applies the threshold of 4 used in the main analysis, and the results are robust to using 3 as a cutoff.

Table 3 cross-classifies feedback by three types. The largest group is non-toxic, negative, and non-constructive (38.5%). An example is *"This is not clear."* Toxic feedback is overwhelmingly negative and non-constructive (8.1%), such as *"You are talking nonsense."* Positive feedback is often non-constructive (76%), usually short affirmations like *"Looks good to me."* Finally, most constructive feedback is non-toxic and negative (70%), providing critical but actionable suggestions. For example, *"Still feels redundant. An angle should just be a union of float and FreeParameter. Once the parameter is bound, you have a new circuit with no free parameters."*.

[Table 3 about here.]

4 Descriptive Facts

This section presents three descriptive facts about code review that motivate the empirical strategy. They highlight who drives variation in feedback dimensions, how feedback is distributed across reviewers and how feedback is structured within teams.

Fact 1: Reviewers Explain Most of the Variation in Feedback Dimensions. To identify the sources of variation in feedback tone and information, I regress predicted feedback scores at the developer–team–week level (continuous measures of toxicity, positivity, and constructiveness) on fixed effects for teams, reviewers, developers, and case-level controls. Figure 3 shows that reviewer fixed effects explain the largest share of variation across all three dimensions. In contrast, team fixed effects account for much less, and developer fixed effects or case characteristics add little explanatory power. These results indicate that variation in feedback primarily reflects reviewer feedback styles rather than differences in team culture, developer traits, or case context.

[Figure 3 about here.]

Fact 2: Feedback Provision is Highly Concentrated. I further examine the distribution of feedback provision and find that it is highly concentrated (Figure 4). A small group of reviewers contributes a disproportionate share of messages in each dimension. The top 5% contribute 60.2% of toxic feedback, 33.8% of positive feedback, and 35.5% of constructive

feedback. This concentration highlights substantial heterogeneity among reviewers in their feedback styles. It provides meaningful variation for the empirical analysis.

[Figure 4 about here.]

Fact 3: Most Teams Follow a Hierarchical Feedback Structure. I measure the hierarchy of a team using the direction of feedback exchanges. A fully hierarchical team has feedback flowing only in one direction, similar to a workplace where managers review juniors but not vice versa. In contrast, reciprocal feedback, where two members review each other's code, indicates a more egalitarian structure. I define the hierarchy score as:

$$\text{Hierarchy Score} = 1 - \frac{2R}{E}$$

where R is the number of reciprocal pairs and E is the total number of feedback messages. A score of 1 indicates purely one-way feedback (fully hierarchical), while values near 0 reflect frequent reciprocity.

Figure 5 shows the distribution of hierarchy scores for teams of size five. Most teams exhibit a hierarchical structure. This means that code reviews tend to flow in one direction rather than occurring between peers. The figure also presents three team feedback networks that illustrate low, medium, and high hierarchy levels. Figure F17 shows that this pattern holds across all teams: in most cases, reviewers specialize in reviewing and do not contribute code themselves.

[Figure 5 about here.]

Taken together, these facts guide the empirical strategy. Feedback tone and information vary primarily across reviewers, not across teams or developers. Feedback provision is concentrated among a small group of reviewers. This combination implies that variation in feedback tone and information, which stems from reviewer styles, generates meaningful differences. I exploit this variation to identify their effects on worker productivity.

5 Empirical Strategy

I estimate how different types of feedback affect developers' future productivity. To illustrate the empirical strategy, I begin with toxic feedback as an example; the same framework applies to the other types.

Consider the following model, where t indexes calendar weeks when feedback is given:

$$Y_{im,[t+1,t+4]} = \beta I^{\text{Toxic}}_{im,t} + \Gamma_t X_{i,t} + \gamma_{m,\text{year}(t)} + \varepsilon_{im,t}, \tag{1}$$

where $Y_{im,[t+1,t+4]}$ is the productivity of developer i in team m during weeks $t+1$ to $t+4$ (e.g., code quantity or quality). The indicator $I^{\text{Toxic}}_{im,t}$ equals 1 if developer i received toxic feedback in team m during week t, and 0 if the feedback was non-toxic. The vector $X_{i,t}$ controls for feedback volume and activity level (measured in deciles) in week $t-1$. I also include team-by-year fixed effects $\gamma_{m,\text{year}(t)}$.[15] The coefficient β captures the effect of receiving toxic feedback in week t.

The main challenge is that OLS estimates of (1) may be biased if the receipt of toxic feedback correlates with unobserved factors that also affect outcomes. This bias could be positive or negative. On the one hand, developers with lower latent ability may be less productive and more likely to receive toxic feedback. Conversely, developers with fewer outside options may interpret toxic feedback as job dissatisfaction and respond by increasing their effort, fearing dismissal. Toxic feedback may also concentrate at specific stages of the project cycle, such as early development, when goals are less defined and productivity more volatile. These unobserved factors complicate identification and can distort OLS estimates in either direction.

The as-if random assignment of developer cases to reviewers (conditional on team-by-year fixed effects) provides variation in toxic feedback. This variation is independent of developer or case characteristics. If reviewers differ systematically in their feedback styles, some are toxic than others. This reviewer feedback style generates variation in developers' exposure to toxic feedback. As a result, the likelihood that a developer receives toxic feedback varies with reviewer assignment, and this variation is independent of the content of the case itself. [16]

To instrument for receiving toxic feedback, I construct a leave-developer-out measure of reviewer toxicity. This follows the judge or officer designs in Dobbie and Song (2015); Dobbie et al. (2017, 2018); Bhuller et al. (2020). Reviewers and developers may interact repeatedly, so a simple leave-one-case-out measure risks bias if the same pair appears elsewhere. I therefore exclude all cases involving the focal developer when computing a reviewer's toxicity rate. Concretely, the instrument is the average share of toxic feedback the reviewer gives in cases with other developers. This procedure ensures that the instrument is not mechanically correlated with the focal developer's outcomes.

$Z^{\text{Toxic}}_{j(i)}$ is the leave-developer-out mean measures of toxic feedback for each reviewer j:

$$Z^{\text{Toxic}}_{j(i)} = \frac{1}{\sum_{h \neq i} n_{hj}} \sum_{h \neq i} D^{\text{Toxic}}_{hj}$$

[15] Following the judge leniency literature, I condition on the block where assignment is as good as random and include block fixed effects. Identification comes from within block comparisons (Dobbie and Song, 2015; Dobbie et al., 2017; Bhuller et al., 2020; Beuschlein, 2024). In our data, reviewer assignment primarily occurs within team-by-year cells. The average reviewer tenure on a team is 1.12 years. I therefore include team-by-year fixed effects. Results are robust to team-by-month fixed effects.

[16] The same logic applies to positive and constructive feedback. Some reviewers are systematically more positive (Figure 4b) or more constructive (Figure 4c). Consequently, the probability that a developer receives a given feedback type depends on the tendencies of the assigned reviewer. I exploit this reviewer-driven variation as the basis for an instrumental variables strategy to identify the causal effects of feedback types.

where D_{hj}^{Toxic} is the number of toxic cases reviewer j gives to developer h, and n_{hj} is the total number of cases reviewer j reviews for developer h.

The main analysis will be based on 2SLS estimates of the second-stage equation (1) and the first-stage for developer i and reviewer j at event time t is given by:

$$I_{im,t}^{\text{Toxic}} = \alpha Z_{j(i)}^{\text{Toxic}} + \delta_t \mathbf{X}_{i,t} + \gamma_{m,\text{year}(t)} + \varepsilon_{im,t}, \tag{2}$$

where the scalar variable $Z_{j(i)}^{\text{Toxic}}$ denotes the toxicity of reviewer j assigned to developer i, and γ represents the impact of reviewer toxicity on the likelihood of receiving toxic feedback. Robust standard errors are clustered at the reviewer level in both stages.[17] I report the robust first-stage F-statistic, which is large in our setting (Staiger and Stock, 1994; Lee et al., 2022).[18]

I interpret the 2SLS estimates within the Local Average Treatment Effect (LATE) framework (Angrist and Imbens, 1994). Under the assumptions of instrument exogeneity and monotonicity, the instrument recovers the local causal effect of receiving toxic feedback among the subgroup of developers whose likelihood of exposure varies with the assigned reviewer. Exogeneity means that reviewer assignment affects developer outcomes only through its impact on targeted feedback type and not through other channels. Monotonicity implies that the instrument affects treatment in only one direction. I provide evidence on the exclusion restriction and monotonicity tests in Section 5.1.

5.1 Assessing the Instrument

Instrument Relevance. Figure 6 provides a graphical representation of the first stage. It shows reviewer-level feedback propensities after residualizing team-by-year fixed effects. To reduce measurement error, I only include teams with at least 2 reviewers and reviewers who have handled at least 10 cases. This selection leaves 480,697 cases and 10,873 reviewers, with an average of 44.2 cases per reviewer. All three panels reveal substantial cross-reviewer variation in the likelihood of sending toxic, positive, or constructive feedback.

Panel (a) in Figure 6 shows that residualized reviewer toxicity. Moving from the 10th to the 90th percentile increases the probability of receiving toxic feedback by 0.2 pp. This is a

[17] I cluster standard errors at the reviewer level because misclassification in feedback detection is likely correlated within reviewers. Each reviewer evaluates many pull requests, which induces within-reviewer correlation in the errors. Clustering at the reviewer level addresses this dependence and yields a valid inference. As a robustness check, Figure F5 reports results with team-level clustering; the estimates are similar.

[18] Assessing the risk of many-weak-instrument bias in judge-style designs remains an open question (Hull, 2017; Frandsen et al., 2023; Bhuller et al., 2020; Agan et al., 2023). In Table D1, I evaluate the robustness of the leave-developer-out reviewer leniency instrument using alternative IV estimators. These include: (i) limited information maximum likelihood (LIML) with all reviewer dummies as instruments; (ii) the modified bias-corrected two-stage least squares (MBTSLS) estimator of Kolesár et al. (2015); and (iii) the unbiased jackknife instrumental variable estimator (UJIVE) of Kolesár (2013). The estimates are quite stable.

26.7% increase relative to the mean toxic feedback rate of 0.76%. For positivity in Panel (b), the shift from the 10th percentile (−0.15) to the 90th percentile (0.16) raises the probability of positive feedback by 31 pp. This represents a 43.9% increase from the mean rate of 70.6%. For constructiveness, moving from the 10th percentile (−0.21) to the 90th percentile (0.20) increases the likelihood of constructive feedback by 41 pp. This is a 70.4% increase over the mean rate of 51.8%.

[Figure 6 about here.]

Table 5 presents first-stage estimates from regressions of indicator variables for receiving each feedback type (toxic, positive, or constructive) on the corresponding reviewer instrument, defined as the individual reviewer's residualized rate of providing the respective feedback type. Column (1) includes team fixed effects, Column (2) replaces these with team-by-year fixed effects, and Column (3) adds controls including developer gender, race, GitHub activity in the year before submission, the first year of GitHub activity, case characteristics, and project stage, defined based on the release version of the team product (See Appendix A.6). Standard errors are clustered at the reviewer level.

Building on these estimates, the coefficients are large and statistically significant across all panels. In Panel (a), a 10 pp increase in a reviewer's residualized toxicity rate raises the probability of receiving toxic feedback by roughly 7 pp. Panel (b) shows a comparable effect for positive feedback (8.2 pp), while Panel (c) reports a slightly larger effect for constructive feedback (8.6 pp).

[Table 4 about here.]

Conditional Independence. A valid instrument must be uncorrelated with developer and case characteristics that influence outcomes. Column (3) of Table 5 adds controls for these predetermined variables to the first-stage regressions. If reviewer assignment is random, the coefficients should remain stable, which is exactly what I find.

As a second test, Figure 7 assesses the randomness of case assignment after controlling for team-by-year fixed effects. In Panel (a), the blue lines with red markers plot coefficients from regressions of reviewer toxicity on standardized developer and case characteristics. None of the coefficients is statistically significant. Similar patterns hold for positive and constructive feedback in Panels (b) and (c). Taken together, these findings provide strong evidence that reviewer assignment is conditionally random.

[Figure 7 about here.]

Exclusion Restriction. Conditional random assignment of cases to reviewers is sufficient to interpret reduced-form effects of reviewer feedback as causal. Interpreting the IV estimates as causal effects of specific feedback types requires the stronger exclusion restriction: feedback must affect developer outcomes only through its designated dimension and not through others.

However, a challenge arises because feedback is multidimensional. A single feedback message may contain several attributes. As Table **??** shows, most toxic feedback is also negative. This overlap raises an identification concern: do the IV estimates capture the effect of toxicity, of negativity, or of both?

I return to this issue after presenting the main results in Section 6.5. There, I show that the estimates are robust to augmenting the baseline model with controls for other reviewer characteristics or by including instruments for alternative feedback dimensions.

Monotonicity Condition. If the causal effect of reviewer feedback were homogeneous across developers, the instrument would need to satisfy only conditional independence and the exclusion restriction. Under heterogeneous treatment effects, recovering the LATE also requires monotonicity—that the instrument affects the probability of receiving a specific type of feedback in only one direction across developers. This assumption implies that developers who would not receive toxic feedback from more toxic reviewers would also not receive toxic feedback from less toxic reviewers.

I cannot directly test this assumption, but I can use several indirect tests common in the literature. Following Bhuller et al. (2020) and Beuschlein (2024), I test monotonicity by checking that first-stage estimates are nonnegative for all subsamples. I split the data by gender, race, activity level, GitHub experience, and follower count (as a proxy for reputation), then re-estimate the first stage for each group. Appendix Table F1 reports the results. Across all splits, the first-stage estimates are positive and statistically different from zero, consistent with the monotonicity assumption.

6 Effects of Feedback on Developer Productivity

This section presents the main results. First, reviewer feedback has a significant impact on subsequent productivity, affecting both the quantity and quality of code. Second, these effects persist over time, affecting developer retention within firms and across the industry. Third, feedback creates spillovers, affecting contributions across teams and coworker communication.

6.1 Main Results

Figure 8 presents 2SLS estimates of the effect of reviewer feedback on developer productivity within a team over the four weeks following a review. To avoid contamination from the review itself, I exclude the current code being reviewed and focus only on new code produced afterwards. Code quantity is decomposed into two components: the extensive margin, indicating whether a developer contributes any code, and the intensive margin, measuring lines of code written conditional on contribution. Code lines include all cases initiated within four weeks after receiving reviewer feedback (see Appendix A.4). Figure 9 summarizes the main results across all three feedback types.

Effects of Toxic Feedback on Productivity. Figure 8a shows that toxic feedback reduces developers' participation by 7%. It also lowers code quantity by 56 log points (43%). Thus, developers are less likely to code. When they code, they produce less.

One possible response to toxic feedback is task substitution: developers may avoid further toxic interactions by shifting from coding to non-code tasks such as answering questions. The estimates, however, show no significant effects on either the extensive or intensive margins of non-code quantity. This rules out substitution. It indicates that toxic feedback lowers overall output rather than redirecting effort toward other activities.

Another hypothesis is that developers respond by producing less but higher-quality code. However, the results do not support this hypothesis. In fact, toxic feedback reduces code quality, as shown by declines in both the share of corrected cases and the share of accepted code. The share of code submitted and accepted in the team code-base decreases by 9.1 pp (13.13%). This suggests that toxic feedback not only reduces how much developers produce but also discourages careful and accurate coding.

A possible alternative explanation for the effect of toxic feedback on developer productivity is that reviewers who provide such feedback affect future task allocation. If reviewers who leave toxic feedback are also less likely to assign work to the same developer again, the drop in output could reflect assignment decisions rather than voluntary adjustment. Figure 9c rules this out. The number of cases assigned to a developer is unaffected by feedback type, suggesting that observed declines reflect behavioral responses by the developer, not reallocation by the reviewer.

[Figure 8 about here.]

Toxic or Negative? Most toxic feedback is negative (85.7%). This raises a key question: are the detrimental effects driven by toxicity itself, or by negativity? Figure 8b shows the effect on developer productivity when receiving respectful criticisms (non-toxic but negative). While toxicity demotivates, respectful criticism feedback does not. In fact, developers rewrite more

lines of code, with only modest quality costs. Toxic feedback harms not because it expresses dissatisfaction, but because it crosses into abuse.

Effects of Positive Feedback on Productivity. Having documented the negative effects of toxic feedback, I now turn to positive feedback. As shown in Figure 8c, positive feedback improves new code quality without slowing quantity. The share of accepted code rises by 4.4 pp (6.35%), indicating that developers maintain their usual pace but devote more effort to accuracy when they feel appreciated.

These findings align with theoretical models of confidence and performance. For example, Compte and Postlewaite (2004) formalize how positive affect and selective recall of successes sustain effort and productivity. Similarly, Bénabou and Tirole (2002) and Kőszegi et al. (2022) emphasize the central role of self-confidence: positive feedback reinforces beliefs about ability and raises effort, while toxic feedback can undermine these beliefs and reduce productivity.

Effects of Constructive Feedback on Productivity. By contrast, constructive feedback has less favorable effects on future productivity. As Figure 8d shows, developers write a similar amount of code in new cases, yet output quality declines: the share of accepted code decreases by 10.39%. Constructive feedback provides actionable information and is often viewed as a form of on-the-job training that builds task-specific human capital. If developers absorb and generalize this feedback, future performance should improve. Instead, the evidence suggests that developers apply the information in the current task but do not carry the lessons forward.[19]

Two mechanisms may explain this pattern. First, the benefits of feedback may accumulate gradually, and may not translate into immediate gains in output quality (Gibbons and Waldman, 2004). Second, developers may face cognitive constraints: they prioritize meeting present demands over integrating lessons into future work (Kahneman, 1973).

Supporting this hypothesis, Figure 9b shows that developers rewrite 49.18% more lines after receiving constructive feedback. Developers shift effort toward revising old code rather than developing new code. This suggests a trade-off between revising old code and developing new code (see Figure 9b and Figure 9a).

These results highlight the negative externality in constructive feedback. Reviewers may not internalize the cost of being overly detailed or even picky. Specific information can force developers to make revisions that satisfy the reviewer but add little to overall quality. Because reviewers do not bear the opportunity cost of developer time, their feedback can inadvertently reduce efficiency. The outcome resembles the dynamics of academic refereeing: excessively picky reports may generate rounds of revision that improve the submitted version but slow

[19] Figure F18 reports results using an alternative threshold (≥ 3) to define constructive feedback. The findings remain similar.

down the production of new and potentially more innovative work.

Long-Run Retention Outcomes. So far, the analysis has focused on short-run productivity. Figures 9d and F21b point to potential longer-run effects on career trajectories, such as whether developers remain in the industry or switch firms within two years after receiving feedback. To examine these outcomes, I link GitHub activity to LinkedIn profiles (Appendix E), applying a conservative matching strategy and manually validating 10% of the links. The preliminary results suggest that exposure to toxic feedback increases the probability of leaving the industry by 3pp and of leaving the firm by 24 pp, though the estimates are not statistically significant. By contrast, positive feedback reduces the probability of industry exit by 9 pp significantly. This is consistent with evidence from Hoffman and Tadelis (2021), who find a positive relationship between the people management skills of managers and employee retention.[20]

Positive feedback significantly increases developers' likelihood of remaining in the industry, suggesting that feedback serves as a micro-level channel through which managerial style and workplace culture influence labor supply in knowledge-intensive sectors. Because employee knowledge is a key asset, high-tech firms view turnover as a major loss and invest heavily in predicting and reducing attrition (Hoffman and Tadelis, 2021). These results suggest that improving feedback practices may be an effective way to strengthen retention and preserve human capital in knowledge-intensive firms.

[Figure 9 about here.]

Interactions Between Feedback Tone and Information. Having shown the effects of each feedback dimension separately, I next examine how feedback tone and information interact. Figure 10 reports the effects on developer productivity. I distinguish between appreciation alone (pink) and appreciation combined with specific and actionable information (teal). The largest gains come from simple appreciation: both code quantity and quality increase.

[Figure 10 about here.]

6.2 Spillover Effects

Spillovers Within Teams? Having shown that feedback affects developer productivity, I next test for spillover effects on coworkers within the same team. Panel (a) of Figure 11 shows that developers who receive positive feedback subsequently send a higher share of positive messages to their coworkers. This result suggests that feedback generates within-team spillovers and can affect the team communication environment.

[20] Hoffman and Tadelis (2021) measure management skills using six survey items, including two directly related to feedback: "communicates clear expectations" and "provides continuous coaching."

Substitution to Other Teams? Beyond within-team spillovers, feedback may also have effects across teams. Two channels are possible. First, feedback may alter developers' overall engagement, leading them to contribute more or less across the platform; in this case, effects should move in the same direction as those within the focal team. Second, feedback may induce substitution, whereby developers reduce output in one team but increase it elsewhere; in this case, effects would move in the opposite direction. Distinguishing between disengagement and reallocation is therefore central to understanding whether feedback changes total output or merely shifts activity across teams.

Panel (b) of Figure 11 shows that positive feedback generates cross-team spillover, increasing code quality in other teams by 5.8%. In contrast, constructive feedback reduces both code quality measures. This pattern is consistent with a trade-off between revising old code and developing new code: when developers receive detailed or demanding feedback, they must devote more time to reworking old code, leaving less time to contribute to other teams.

Figure F20 shows no significant change in developers' code quantity or non-code quantity in other teams after receiving feedback in their focal team.

Overall, positive feedback not only increases developers' own productivity but also leads them to send more positive messages to their coworkers, fostering a better team environment and generating productivity gains across the platform.

[Figure 11 about here.]

6.3 Comparison to OLS

Table 6 compares the OLS and IV estimates. The OLS results are from equation (1). Panel A shows that OLS suggests developers who receive toxic feedback are more productive: code output rises by 6.72% and non-code activity by 8.33%, though code quality declines. The IV estimates reverse this narrative. Toxic feedback reduces output on all margins: code quantity falls by 42.88% and output code by 13.13%. The difference underscores the endogeneity in OLS. First, more productive developers both produce more output and receive more reviews, which increase their likelihood of receiving feedback, including toxic feedback. Second, reviewers may direct feedback to developers with fewer outside options, who may respond by increasing effort. Both factors make toxic feedback appear beneficial in OLS, although the underlying causal effect is negative according to IV estimates.

Panel B shifts the focus to positive feedback. OLS estimates show increases in code quantity (19.5 percent) and non-code activity (14.7 percent). The IV results tell a different story: positive feedback reduces non-code activity by 13.9 percent while improving code quality by 6.4 percent. The difference arises from selection. High-performing developers are more likely to receive positive feedback because they already produce good work. As a result, the OLS

correlation overstates the benefits of positive feedback. Once reviewer assignment is exogenous, the quality improvement remains, but the spurious association with output disappears.

Panel C turns to constructive feedback. OLS estimates indicate a substantial increase in code quantity and a minor decrease in code quality. In contrast, IV estimates indicate muted quantity effects and a larger decrease in code quality, up to 10.39%. Developers who are already productive are more likely to receive constructive suggestions for improvement, which makes OLS estimates upward biased. Once reviewer selection is removed, the true short-term effect of constructive feedback is negative.

Taken together, these results indicate that OLS systematically overstates the benefits of feedback due to endogenous reviewer assignment. After correcting for these biases, IV results reveal that feedback tone and information affect productivity. These average effects may mask important differences across developers and teams. I next examine how feedback effects vary by demographics, ability, tenure, and team structure.

[Table 5 about here.]

6.4 Heterogeneous Effects

I explore how feedback effects differ across subgroups, focusing on differences by demographics, coding ability, prior exposure, tenure, team structure, and the reviewer–developer match.

By Race. Figure F9 compares Asian and White developers, the two largest groups in the data (see Appendix A.2 for race prediction). Most feedback types yield similar effects across groups, but responses to toxic feedback differ: Asian developers increase future code output, while White developers decrease. This may reflect different cultural norms around blunt critique and parenting styles (e.g., greater tolerance for harsh feedback, or "tiger parenting"), with Asian developers more likely to respond with extra effort, whereas White developers respond by disengaging (Kim et al., 2013; Gelfand et al., 2011; Pinquart and Kauser, 2018).

By Gender. Figure F10 reports feedback effects by developer gender (see Appendix A.1 for gender prediction). Male and female developers respond similarly to positive and constructive feedback, but differ on toxic feedback: males reduce code quantity, while females slightly increase it, though this is not statistically significant. With women making up only 12% of the sample, this subgroup is likely selected. Future research should examine the mechanisms behind these differences.

By First vs. Repeat Exposure. Figure F11 compares developers' responses to their first exposure to each feedback type versus repeated exposures. Most patterns are similar across exposures, suggesting developers do not quickly adapt to feedback. The main exception is positive feedback: improvements in code quality appear mainly after repeated exposures, consistent with developers internalizing appreciation more strongly over time.

By Developer Ability. Figure F12 compares feedback effects for developers with high and low prior ability, defined by whether pre-feedback code quality is above or below the median. Toxic feedback reduces code quality similarly for both groups. Positive feedback slightly benefits high-ability developers, while constructive feedback tends to lower output for low-ability developers, possibly because detailed suggestions overwhelm those with less skill.

By New Hires vs. Incumbents. Figure F13 compares developers who have just joined a team with those who have been on the team longer. New hires are defined as developers within their first two months on a team; all others are classified as incumbents. Overall, 14.2% of developers are new hires.[21] While incumbents represent a selected group who have chosen to stay, the key distinction lies between developers unfamiliar with the team's culture and those already integrated. Toxic feedback is highly damaging for incumbents, whereas positive feedback most strongly improves their quality. The contrast suggests that feedback becomes more consequential once developers have integrated into the team: incumbents internalize praise and criticism more strongly, while new hires treat feedback as part of the learning process.

By Team Hierarchy. Figure F14 examines feedback effects by team hierarchy score. In more hierarchical teams, toxic feedback reduces future code quality more, while positive feedback improves it more, possibly because appreciation from high-status reviewers has stronger motivational effects.

By Demographic Matching. Figure F15 and Figure F16 show feedback effects vary with reviewer–developer demographic match. Toxic feedback is more damaging within-group, but neutral or mildly positive across groups. Positive feedback gives small gains for most, while constructive feedback generally lowers future code quality. These patterns suggest demographic similarity affects feedback interpretation, especially for negative feedback.

6.5 Threats to Exclusion Restriction

Interpreting the IV estimates as causal requires that each type of feedback affects outcomes solely through its designated channel. However, two concerns threaten this exclusion restric-

[21] If I use the first week or first month to define new hires, the share becomes only 4% or 8%.

tion. First, feedback is multidimensional. If the instruments influence multiple dimensions at once, the exclusion restriction may not hold. Second, reviewer characteristics such as gender, seniority, or engagement may correlate with the feedback provided. As a result, the estimated effects may reflect reviewer traits rather than the content of the feedback.

To address these issues, I first test whether each instrument affects only the targeted type of feedback. Then, I control for observable reviewer characteristics and demonstrate that the main results remain robust.

6.5.1 Assessing Instrument Specificity Across Feedback Types

To evaluate whether the exclusion restriction holds, I first examine whether each instrument shifts only its intended feedback type or also affects other dimensions.

These concerns apply to all types of feedback. For example, suppose the instrument for toxic feedback also shifts the chance of receiving positive or constructive feedback. In that case, the estimated effect of toxicity includes the influence of encouragement or detailed suggestions. Likewise, if the instrument for positive feedback alters the probability of receiving constructive or toxic feedback, the estimate cannot isolate the effect of positivity. In each case, the exclusion restriction requires that the instrument move only one dimension of feedback.

To formalize this, I extend the baseline IV model in equations (1) and (2), separating the three types of feedback:

$$Y_{im,[t+1,t+4]} = \beta^{\text{Toxic}} I^{\text{Toxic}}_{im,t} + \beta^{\text{P}} I^{\text{P}}_{im,t} + \beta^{\text{C}} I^{\text{C}}_{im,t} + \Gamma_t X_{i,t} + \gamma_{m,\text{year}(t)} + \varepsilon_{im,t}, \tag{3}$$

$$I^{\text{Toxic}}_{im,t} = \phi^{\text{Toxic}} Z^{\text{Toxic}}_{i,t} + \delta^{\text{Toxic}} X_{i,t} + \gamma^{\text{Toxic}}_{m,\text{year}(t)} + \nu^{\text{Toxic}}_{im,t}, \tag{4}$$

$$I^{\text{P}}_{im,t} = \phi^{\text{P}} Z^{\text{P}}_{i,t} + \delta^{\text{P}} X_{i,t} + \gamma^{\text{P}}_{m,\text{year}(t)} + \nu^{\text{P}}_{im,t}, \tag{5}$$

$$I^{\text{C}}_{im,t} = \phi^{\text{C}} Z^{\text{C}}_{i,t} + \delta^{\text{C}} X_{i,t} + \gamma^{\text{C}}_{m,\text{year}(t)} + \nu^{\text{C}}_{im,t}, \tag{6}$$

Let $Y_{im,[t+1,t+4]}$ denote the productivity of developer i in team m during weeks $t+1$ to $t+4$. The variables $I^{\text{Toxic}}_{im,t}$, $I^{\text{P}}_{im,t}$, and $I^{\text{C}}_{im,t}$ are indicators equal to 1 if developer i received toxic, positive, or constructive feedback, respectively, in team m during week t; each equals 0 if the feedback was non-toxic, negative, or non-constructive. The control vector $\mathbf{X}_{i,t}$ includes variables such as the deciles of feedback volume and activity level in week $t-1$. I also include team-by-year fixed effects $\gamma_{m,\text{year}(t)}$, and the error term $\varepsilon_{im,t}$.

The augmented IV system in equations (3)–(6) addresses potential bias in the baseline IV

estimates. The concern is that the instrument for toxic feedback, $Z_{i,t}^{\text{Toxic}}$, may be correlated with the receipt of other types of feedback—such as positive ($I_{im,t}^{\text{P}}$) and constructive ($I_{im,t}^{\text{C}}$) feedback—even after conditioning on their instruments $Z_{i,t}^{\text{P}}$ and $Z_{i,t}^{\text{C}}$. If positive and constructive feedback also affect developer productivity, conditional on toxic feedback and controls, then the exclusion restriction is violated. In this case, $Z_{i,t}^{\text{Toxic}}$ may influence $Y_{im,t}$ not only through toxic feedback, but also indirectly by shifting exposure to other types of feedback.

Figure F6 presents estimation results from equations (3)–(6) for the key outcome variables. Comparing Figure F6 with the corresponding results in Figure 8 shows that the estimates are nearly identical.

6.5.2 Accounting for Reviewer Characteristics in Feedback Effects

A second concern for causal interpretation is that the estimated effects of feedback may be confounded by underlying reviewer characteristics. That is, feedback content may be correlated with reviewer traits that independently influence team outcomes. For example, if male or more senior reviewers tend to provide more toxic feedback, the estimated impact of toxicity may simply reflect the effects of reviewer gender or experience.

To address this concern, I augment the baseline specification by controlling for standardized reviewer characteristics, including gender, race, seniority (measured by the first year of GitHub activity), number of followers, and recent reviewing activity. As shown in Figure F7, the results remain stable, suggesting that the estimated feedback effects are not driven by reviewer demographics or experience but instead reflect the content of the feedback itself.

6.6 Unobserved Offline Feedback

The analysis so far considers only feedback delivered on the GitHub platform. Yet reviewers and developers may also interact offline, and such unobserved exchanges could bias the estimated effects. Consider toxic feedback. If reviewers are toxic both online and offline, the omission of offline interactions leads the estimated on-platform effect $\hat{\beta}_t^{\text{online}}$ to be overstated, since it absorbs both channels. If instead reviewers moderate their online feedback for reputational reasons and redirect toxicity offline, the on-platform estimate understates the overall effect. The same logic applies to positive and constructive feedback: whether offline interactions reinforce or substitute for online ones determines whether the measured effect is too large or too small.

Multi-located vs. Co-located Teams. To assess the scope of bias from offline feedback, I compare multi-located and co-located teams. Geographic dispersion of team members raises barriers to in-person communication, making feedback in multi-located teams more likely to

occur online. I therefore interact feedback types with a co-location indicator C_{im}. The baseline effect β_t^θ reflects multi-located teams, where feedback is primarily online. The effect in co-located teams is $\beta_t^\theta + \eta_t^\theta$, with η_t^θ capturing how offline interaction shifts the impact relative to the online-only baseline.

$$
\begin{aligned}
Y_{im,[t+1,t+4]} &= \beta_t^\theta \, I_{im,t}^\theta + \eta_t^\theta \left(I_{im,t}^\theta \times C_{im} \right) + \mathbf{X}_{i,t}' \Gamma_t + \gamma_{m,\text{year}(t)} + \varepsilon_{im,t} \\
I_{im,t}^\theta &= \pi_t^\theta \, Z_{im,t}^\theta + \Gamma_t^I \mathbf{X}_{i,t} + \gamma_{m,\text{year}(t)} + u_{im,t}^I \\
\left(I_{im,t}^\theta \times C_{im} \right) &= \lambda_t^\theta \left(Z_{im,t}^\theta \times C_{im} \right) + \Gamma_t^{IM} \mathbf{X}_{i,t} + \gamma_{m,\text{year}(t)} + u_{im,t}^{IC}
\end{aligned}
$$

I classify a team as co-located if all members are based in the same country (28.17% of teams meet this definition).[22] Table F2 reports the interaction estimates η^θ. Most are statistically insignificant, except the intensive margin of non-code quantity under constructive feedback. Overall, these results suggest that unobserved offline communication introduces little bias into the estimated effects of feedback.

7 Reviewer Quality

Having established the causal effects of feedback on worker outcomes, I turn to its role in explaining manager quality. A central challenge in measuring manager quality using worker productivity value-added (VA) is sorting. If more productive workers are systematically matched with certain managers, estimated manager VA may reflect differences in worker ability rather than the causal effect of managers on productivity. The same concern arises in the teacher VA literature, where students of different abilities may be non-randomly assigned to particular teachers (Rothstein, 2010, 2017).

The ideal design would involve randomly assigning many workers to each manager. This way, manager quality could be defined as the systematic change in worker productivity after meeting a manager. In this framework, good managers are those who consistently raise productivity relative to a worker's baseline.[23]

Because random assignment is rarely available, researchers have developed alternative approaches. One strand estimates manager fixed effects in regressions of firm or worker outcomes.

[22] Location is inferred from developers' self-reported profiles; see Appendix A.3. Observations with missing location are excluded when constructing the variable but retained in regressions. As a more flexible alternative, I measure dispersion by identifying the country with the largest share of members and calculating that share. Teams with a share below the sample median are classified as multi-located, yielding a more balanced division between co-located and dispersed teams. Figure F19 reports results from separate regressions in each subsample.

[23] I abstract from managers' role in selecting people for their teams. As Hoffman and Tadelis (2021) discuss, good people managers may excel at identifying talent. Here, I focus only on how much managers change workers' productivity before and after the match, ruling out the possibility that better managers select better workers.

These often exploit workers who switch managers (e.g., Bertrand and Schoar, 2003; Lazear et al., 2015; Bender et al., 2018; Diaz et al., 2025). Another strand measures manager quality using survey responses (e.g., Ichniowski et al., 1997; Bloom and Van Reenen, 2007; Hoffman and Tadelis, 2021). Although these approaches have produced valuable insights into the role of managers, it is important to note that fixed-effects methods depend on worker mobility.[24] Meanwhile, survey-based measures reflect perceptions that may only imperfectly capture objective effects on productivity.

A related literature in education evaluates teacher quality using VA methods. These methods measure teachers by the test score gains of their students (Hanushek, 1971; Kane and Staiger, 2008; Chetty et al., 2014) This framework is attractive because it leverages the performance of all students, rather not only those who switch teachers. It relies on objective outcome measures. Building on this work, I adopt the forecast-based estimator of Chetty, Friedman, and Rockoff (2014). This method addresses sorting concerns and improves estimation precision. The idea is to forecast a manager's value-added in period t using data from other periods. It applies jackknifing and ex-ante shrinkage to reduce noise and bias. This estimator sharpens inference by separating actual manager effects from random fluctuations in worker performance.

7.1 Reviewer Quality Measures: Value-Added (VA)

I first use the forecast-based estimator to estimate reviewer VA in a random-assignment sample of developers and reviewers, where sorting is absent. Reviewer quality is measured as the change in developer productivity before and after meeting the reviewer. I then extend the estimator to the full sample, which includes both *random* and *non-random* assignment. To validate the approach, I compare overlapping reviewers across *random* and *non-random* assignment samples. The estimates align closely, providing evidence that the forecast-based estimator generates valid measures of manager quality even outside the random assignment.

I apply the forecast-based estimator in three steps. First, I compute residuals of developer code output after controlling for observables; these residuals capture reviewer contributions and estimation noise. Second, I estimate the best linear predictor of each reviewer's mean residual in period t using their residuals from all other periods. Third, I form the reviewer's VA in period t by forecasting their residuals with data from every other period. This procedure jackknifes across periods, incorporates shrinkage weights, and allows reviewer VA to vary over time.[25] Appendix D.2 provides further details.

Developer weekly code output is measured by the total number of code lines they submitted

[24] See, for example, Abowd, Kramarz, and Margolis (1999); Andrews, Gill, Schank, and Upward (2008); Bonhomme, Lamadon, and Manresa (2019).

[25] This approach ensures that VA estimates are not mechanically correlated with the outcome being validated and reduces noise, which is particularly important when sample sizes are small (Bacher-Hicks and Koedel, 2023).

to a team in a week. I control for the code-case characteristics (e.g., bug fix, enhancement), team- and firm-level code output, and demographics.[26] Following prior work, I restrict to cases where prior output is available and exclude teams with fewer than 10 members.

Panel (a) in Figure 12 plots the empirical distribution of reviewer VA estimates in the *random sample*. A one standard deviation (SD) increase in reviewer VA increases developer code quantity by 0.20 SD and code quality by 0.10 SD. Panel (b) shows that reviewer VA estimates are strongly correlated across these two outcomes, suggesting that reviewers who increase code quantity also tend to improve code quality. Figure F2 shows the correlation between a developer's weekly code quantity and code quality. This relationship is positive both with and without developer fixed effects.

[Figure 12 about here.]

I extend the analysis to the full sample, which includes teams where reviewer assignment is not random. Panel (c) and (d) in Figure 12 compares the distribution of reviewer VA estimates in the random-assignment sample and the full sample. The two distributions are very similar across both outcome measures of developer productivity, suggesting that the forecast-based estimator produces consistent results beyond the random-assignment setting. Figure F1 shows that reviewer VA estimates are strongly correlated across these two outcomes in the full sample.

To validate the forecast-based approach, I focus on reviewers who appear in both the random and non-random samples. Since these reviewers work with different developers across samples, I compare their relative rankings across estimation methods. Figure 13 shows that reviewer VA estimates are highly correlated across samples (correlation = 0.7).

[Figure 13 about here.]

Firms devote substantial resources to recruiting and retaining high-quality managers (e.g., Lucas Jr, 1978; Bloom et al., 2013; Gabaix and Landier, 2008). I have shown that managers' feedback affects worker productivity. To connect these results, I next estimate the effect of being assigned to a high-VA reviewer on developer output:

$$Y_{imt} = \beta \, \text{HighVA}_{j(i)t} + \mathbf{X}'_{it}\boldsymbol{\beta_2} + \gamma_{m,\text{year}(t)} + \varepsilon_{im,t}, \tag{7}$$

[26] In teacher value-added (VA), teachers are assigned at the start of year t, and researchers compare end-of-year scores in t with baseline scores in $t-1$. In my setting, reviewers are assigned in week t, but code produced in week t precedes the review. I therefore compare output in $t+1$ to output in $t-1$, using week t only for assignment. In the estimation, I winsorize code output at the 95th percentile to address unusually large code commits and then standardize the log of code lines. I also include a control for missing code output using an indicator variable.

where Y_{imt} is developer i's code output in team m at week t, $\text{HighVA}_{j(i)t}$ equals one if the assigned reviewer j has above-median reviewer value-added (VA), $\mathbf{X}_{it}$ is a vector of developer-level controls, and $\gamma_{m,\text{year}(t)}$ are team-by-year fixed effects.

I find that developers assigned to high-VA reviewers produce 53% more code than those reviewed by lower-VA reviewers. However, these gains can be offset by the tone of feedback. As shown in Figure 8a, a single toxic feedback message reduces subsequent code quantity by about 43%. Taken together, these results suggest that while reviewer skill increases developer productivity, toxic feedback can nearly eliminate the benefits of high managerial quality.

7.2 Explanatory Power of Feedback for Reviewer VA

VA measures capture the overall impact of reviewers on worker outcomes. These estimates are constructed without incorporating any information about feedback content. I then ask: How much of this variation in reviewer VA can be explained by the feedback they provide?

To assess this explanatory power, I use Extreme Gradient Boosting, a flexible ensemble method that captures nonlinearities and interactions and is increasingly applied in economics (e.g., Zeltzer et al., 2023; Nekoei et al., 2024). Appendix G provides details.

I first include reviewer characteristics, such as gender and race, to provide a benchmark for the explanatory power. I then compare three sets of feedback-based covariates. The first set includes only feedback quantity, measured by the number of messages sent. The second adds the full text of feedback, represented by 768-dimensional semantic embeddings. The third uses selected feedback features, defined as the shares of feedback that are toxic, positive, or constructive.

Figure 14 provides clear evidence that feedback substantially explains variation in reviewer VA. For the random-assignment sample, reviewer gender and race explain just 0.21% of VA variance, while feedback quantity accounts for about 7%. Incorporating semantic embeddings greatly increases explanatory power to nearly 22%. Using three targeted feedback features captures about 84% of the full model's power. Panel (b) shows similar patterns for the full sample.

These results indicate that feedback can explain a large share of manager quality. This suggests that firms can evaluate managers through the feedback they provide in real time. This offers a practical tool for identifying high-quality managers.

[Figure 14 about here.]

8 Conclusions

Managers play a central role in affecting worker outcomes. Yet the mechanisms through which they affect remain less understood. This paper opens that "black box" by examining how managers affect workers through feedback. Using over 230 million code review messages from software teams on GitHub, I exploit random reviewer assignment to show that feedback has large and significant effects on productivity and retention. Toxic feedback reduces future productivity. Respectful criticism (negative but non-toxic) has no such detrimental effects. Positive feedback generates gains in productivity, retention, and spillovers.

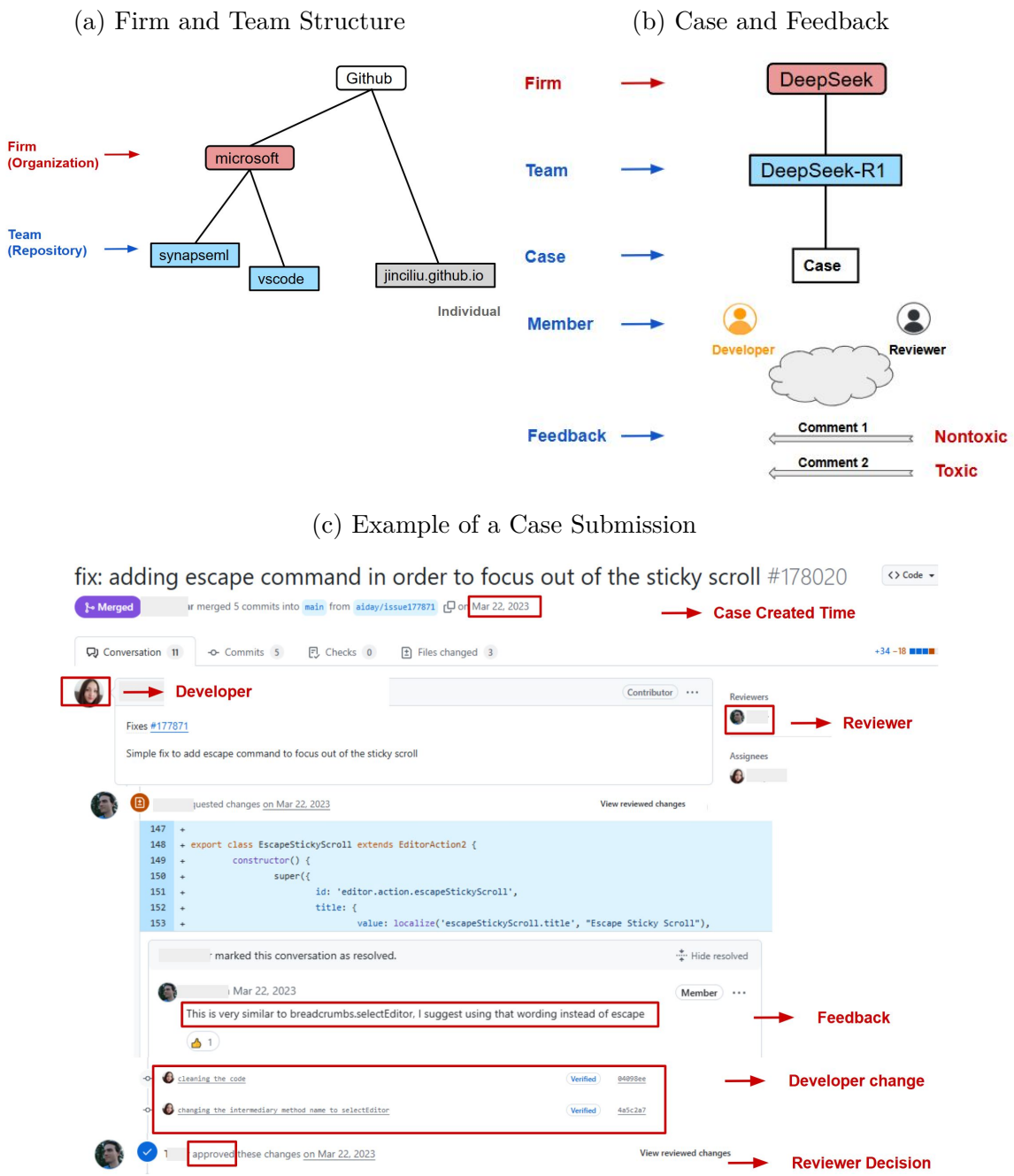
Beyond affecting worker outcomes, I show that feedback explains a sizable share of the variation in manager quality. I use a value-added framework to measure manager quality and find that feedback explains a substantial share of this variation. In contrast, manager demographics account for very little.

Toxic feedback is costly. Small improvements can pay off at scale. The GitHub ecosystem is valued at about $8 trillion. Within this economy, toxic feedback reduces output by 43%. Making feedback more positive raises code quality by about 6%. It also generates peer spillovers of roughly 7%. These results imply that even modest improvements in tone could produce sizable productivity gains globally.

Feedback is more than information transfer. Its tone also affects motivation and productivity. Firms that monitor and improve feedback can increase productivity, strengthen retention, and improve workplace culture. The implications of feedback extend beyond the workplace. Teachers, parents, and academic reviewers rely on feedback to guide and motivate students, children, and researchers. Understanding the effects of feedback matters not only for organizations but also for education, families, and innovation.

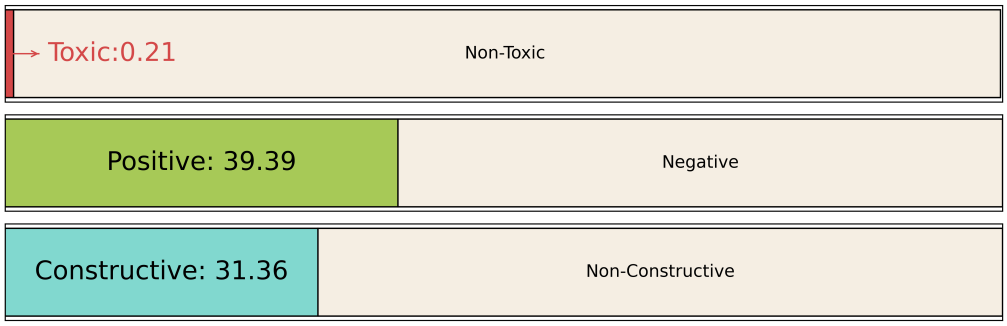
Methodologically, the paper shows that large language models can classify feedback at scale. This enables systematic study of managerial communication across settings. Future research could extend this framework to peer feedback and upward feedback from developers to reviewers. It could also examine heterogeneity across different types of developers. More broadly, studying how different forms of communication affect overall team performance would help identify practices that make teams more productive in modern workplaces.

Figure 1: Organization Structure and Feedback on GitHub



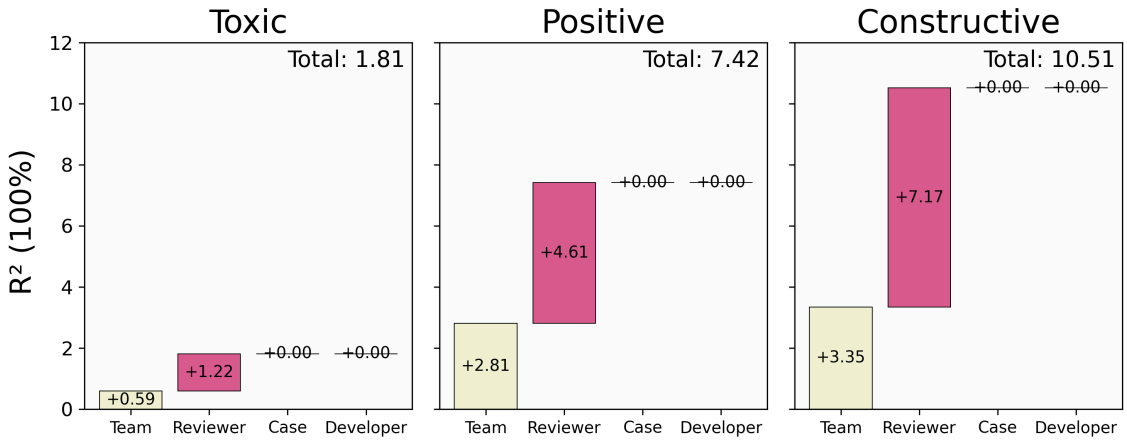
Notes: **Panel (a)** shows that some teams (repositories) belong to firms (organizations), while others do not. In this study, I include only teams affiliated with firms, excluding those created for individual or hobby purposes such as personal websites. **Panel (b)** illustrates within-team communication: after a developer submits a case, reviewers provide feedback on the submission. **Panel (c)** presents an example of a real case (Pull Request #178020), showing the added code and the subsequent reviewer–developer interaction.

Figure 2: Feedback Classification Results



Notes: This figure shows the share of feedback messages by type in the random reviewer sample: 0.21% toxic, 99.79% non-toxic, 39.40% positive, 60.60% negative, 31.36% constructive and 68.64% non-constructive. These labels are classified using LLMs (BERT-based models and GPT) that analyze feedback, including toxicity classification flags for intentional harm language, positivity analysis to classify positive or negative tone, and constructiveness classification to evaluate whether feedback offers specific, actionable information.

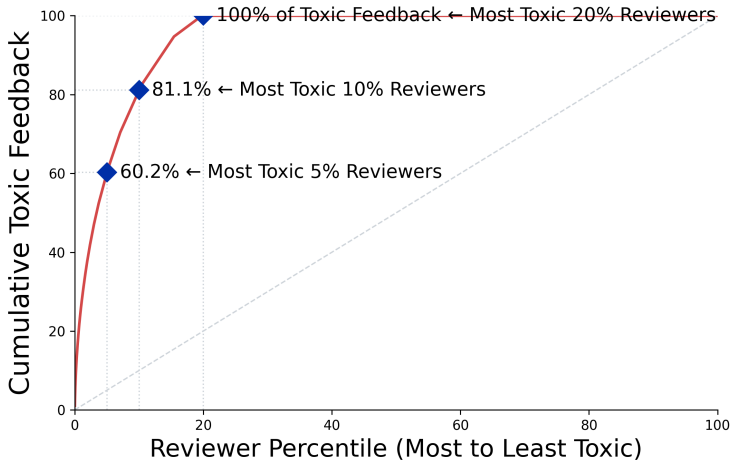
Figure 3: Sources of Variation in Feedback Tone and Information



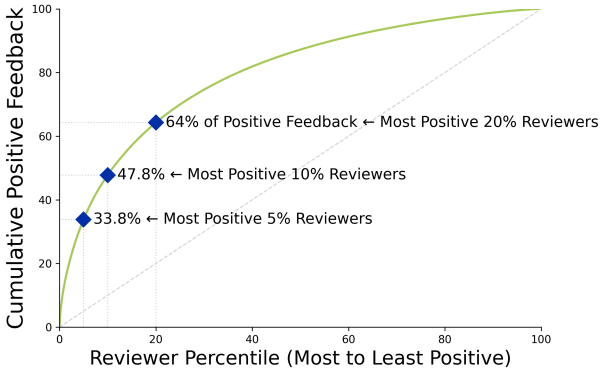
Notes: This figure decomposes variation in feedback tone (toxicity and positivity) and information (constructiveness). Each bar shows how much of the variation in weekly feedback shares is explained by different sources. The unit of observation is a developer–week–team. The dependent variable is the share of feedback messages that are toxic, positive, or constructive, measured within each team week. Explanatory factors include team, reviewer, and developer fixed effects, and case characteristics such as whether the case involves an enhancement or a bug. R^2 values are shown in percentage terms and stacked to illustrate cumulative explanatory power. The sample is restricted to the random reviewer subsample.

Figure 4: Unequal Distribution of Feedback across Reviewers

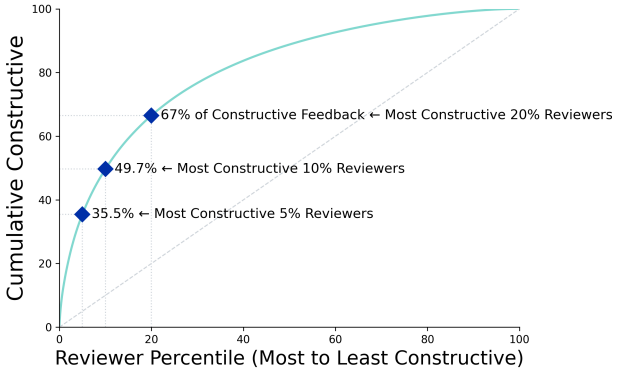
(a) Toxic Feedback



(b) Positive Feedback

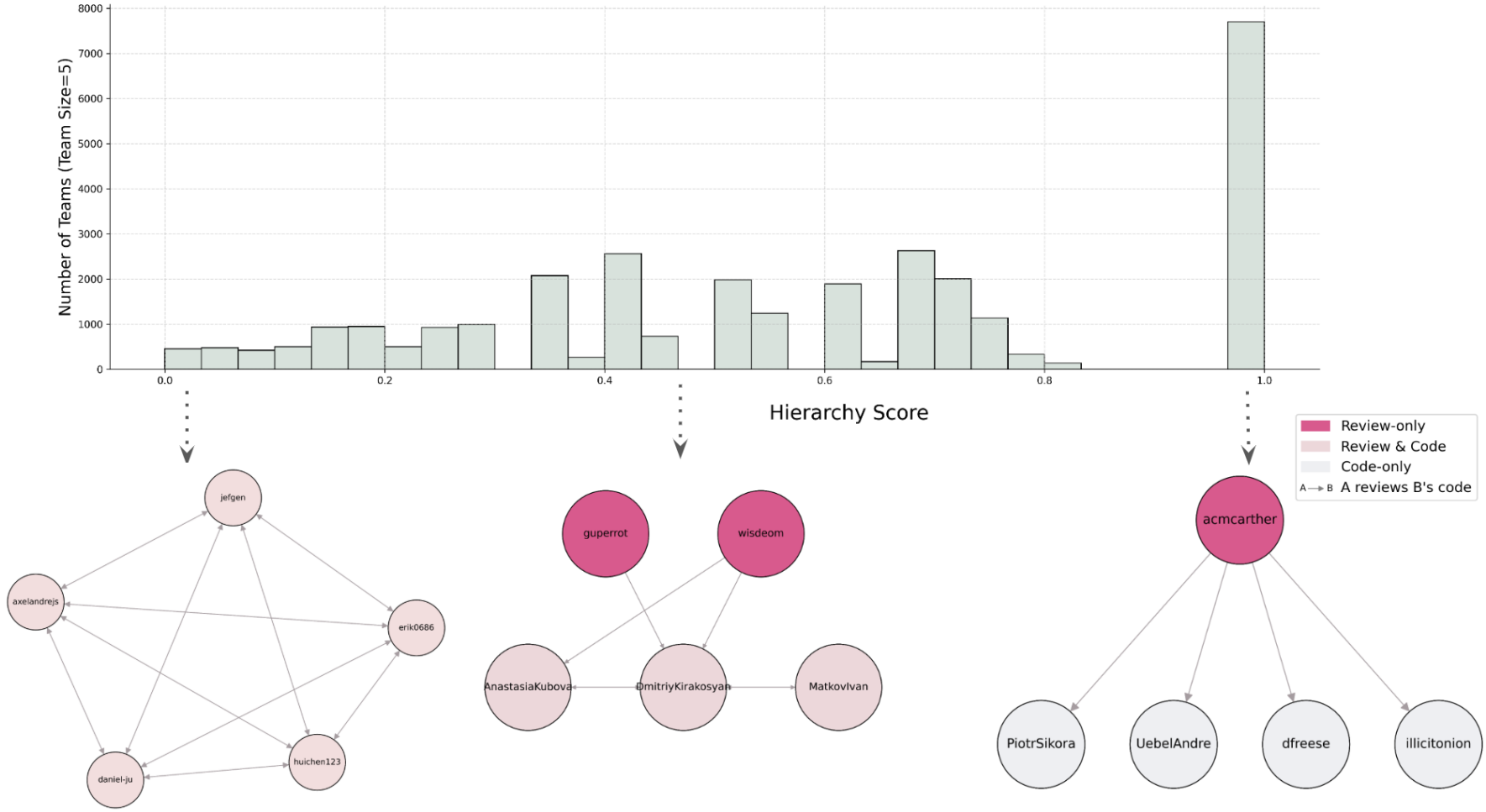


(c) Constructive Feedback



Notes: Each panel plots the cumulative distribution of feedback messages across reviewers, ranked from most to least active in each feedback dimension. The x-axis shows the cumulative share of reviewers, and the y-axis shows the cumulative share of messages of that type. The unit of analysis is the reviewer. **Panel (a)** shows toxic feedback, **Panel (b)** positive feedback, and **Panel (c)** constructive feedback. A steeper curve indicates greater concentration, meaning a smaller share of reviewers accounts for most messages of that type. The sample is restricted to the random reviewer subsample.

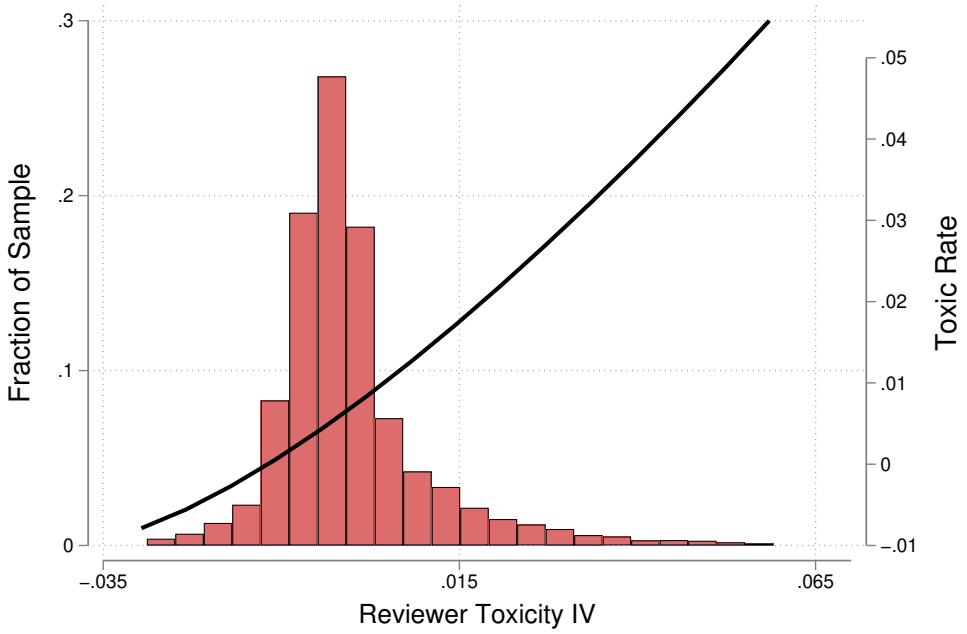
Figure 5: Distribution of Hierarchy Scores and Example Team Networks



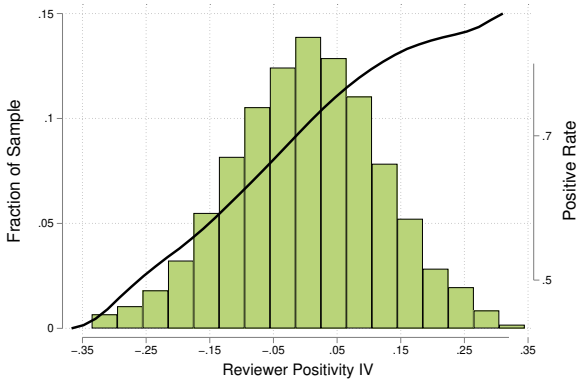
Notes: The top panel shows the distribution of hierarchy scores among teams with five members. The hierarchy score measures how one-directional feedback flows within a team: a score of 1 indicates fully hierarchical feedback (one-way), while values near 0 indicate reciprocal reviewing. Teams with only one case are excluded. The bottom panel shows three examples of team feedback networks. In calculating the hierarchy score, all feedback instances are counted, but duplicate links (when a reviewer evaluates multiple cases for the same developer) are collapsed into a single edge for clarity. The left network has a score of 0, where members review each other's code; the middle network has a score of 0.5, indicating partial reciprocity; and the right network has a score of 1.0, indicating complete hierarchy. Node colors denote roles (review-only, mixed, or code-only). Directed edges represent review relationships, where A → B means A reviews B's code. Figure F17 extends this analysis to the team–year level and shows that most teams have hierarchy scores near 1, suggesting predominantly hierarchical feedback structures.

Figure 6: Distribution of Reviewer Instrument Variables and First Stage

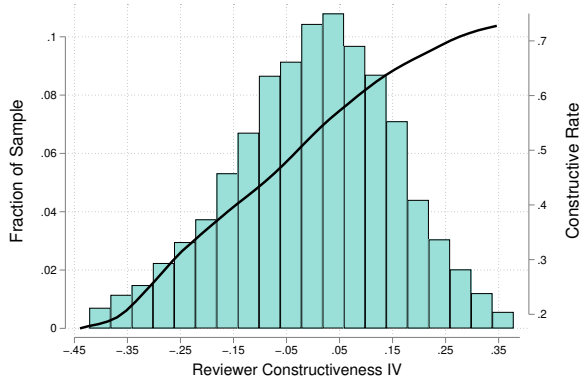
(a) Reviewer Toxicity



(b) Reviewer Positivity



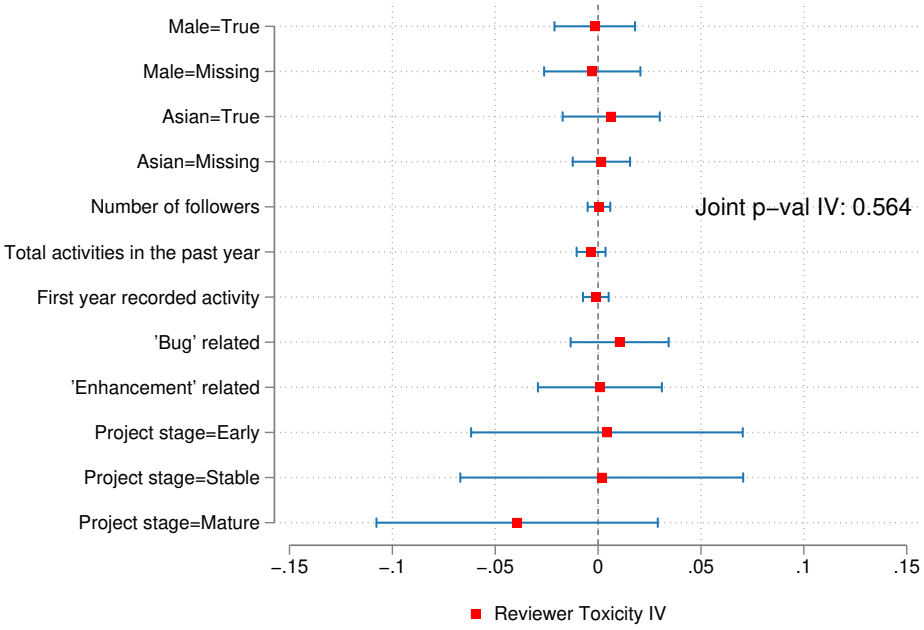
(c) Reviewer Constructiveness



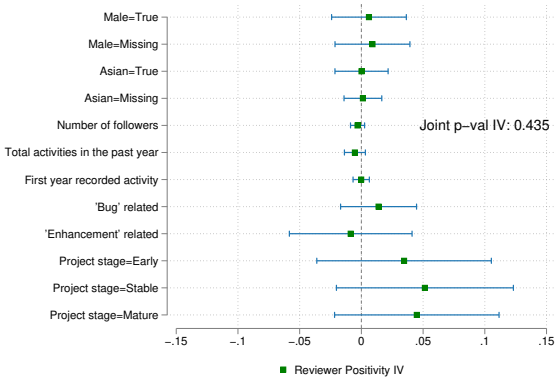
Notes: Each panel shows the distribution of reviewer instruments and the corresponding first-stage relationship between the reviewer's feedback style and the feedback received by developers. The instrument is the reviewer's leave-developer-out average feedback share, residualized by team-by-year fixed effects. I only include teams with at least 2 reviewers and reviewers who have handled at least 10 cases. This leaves 480,697 cases and 10,873 reviewers, with an average of 44.2 cases per reviewer. The bars represent the distribution of reviewer instrument values, while the solid line shows the kernel-weighted local polynomial fit of the first stage. **Panel (a)** plots reviewer toxicity (top and bottom 1% excluded), **Panel (b)** positivity, and **Panel (c)** constructiveness.

Figure 7: Balance of Reviewer Instruments

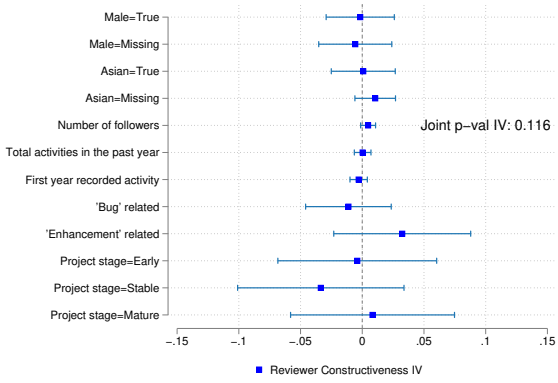
(a) Reviewer Toxicity



(b) Reviewer Positivity



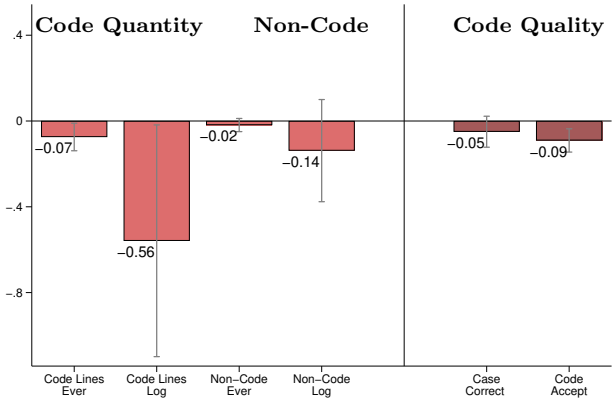
(c) Reviewer Constructiveness



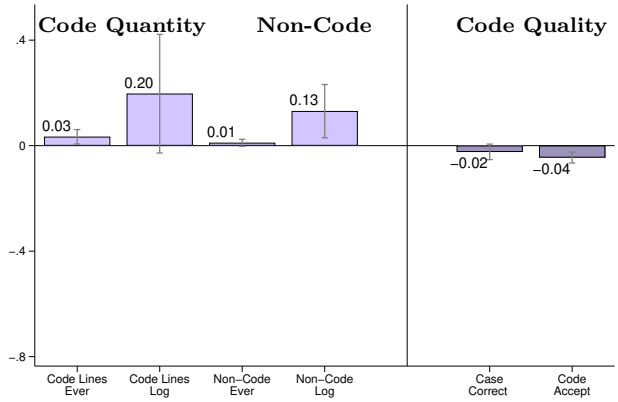
Notes: Each panel reports OLS coefficients from regressions of the reviewer instrument on standardized developer and case characteristics. The instrument is the reviewer's leave-developer-out average feedback share residualized by team-year fixed effects. Developer characteristics include gender, race, number of followers in 2023, prior-year GitHub activity, and first year of GitHub activity. Case characteristics include labels such as Bug and Enhancement and project stage constructed from the team's release version. Coefficients are shown with 95% confidence intervals. p-values shown in the panels refer to joint significance tests of all characteristics. Standard errors are clustered at the reviewer level. The sample is the random-assignment subsample. Appendix A.1 and Appendix A.6 provide variable definitions.

Figure 8: Effect of Feedback on Developer Productivity within a Team (Next 1–4 Weeks)

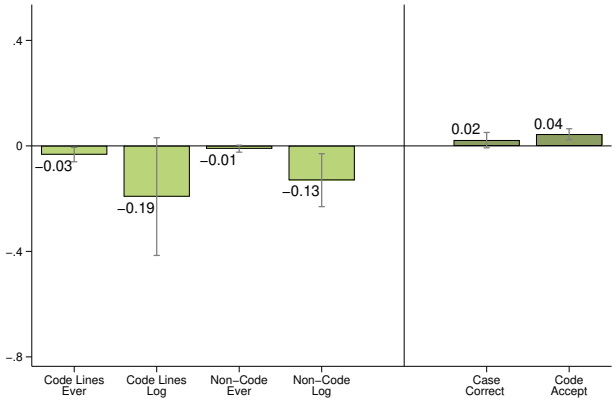
(a) Toxic Feedback



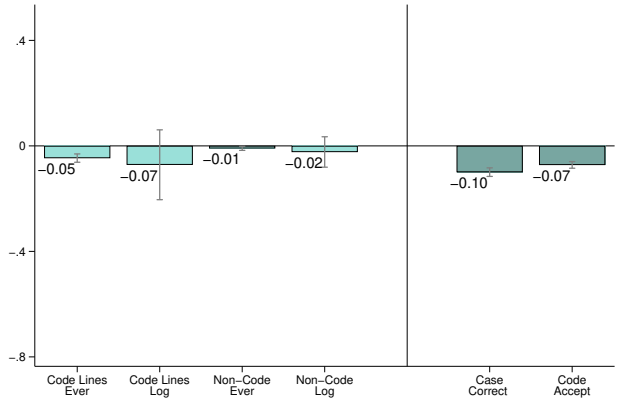
(b) Negative & Non-Toxic Feedback



(c) Positive Feedback

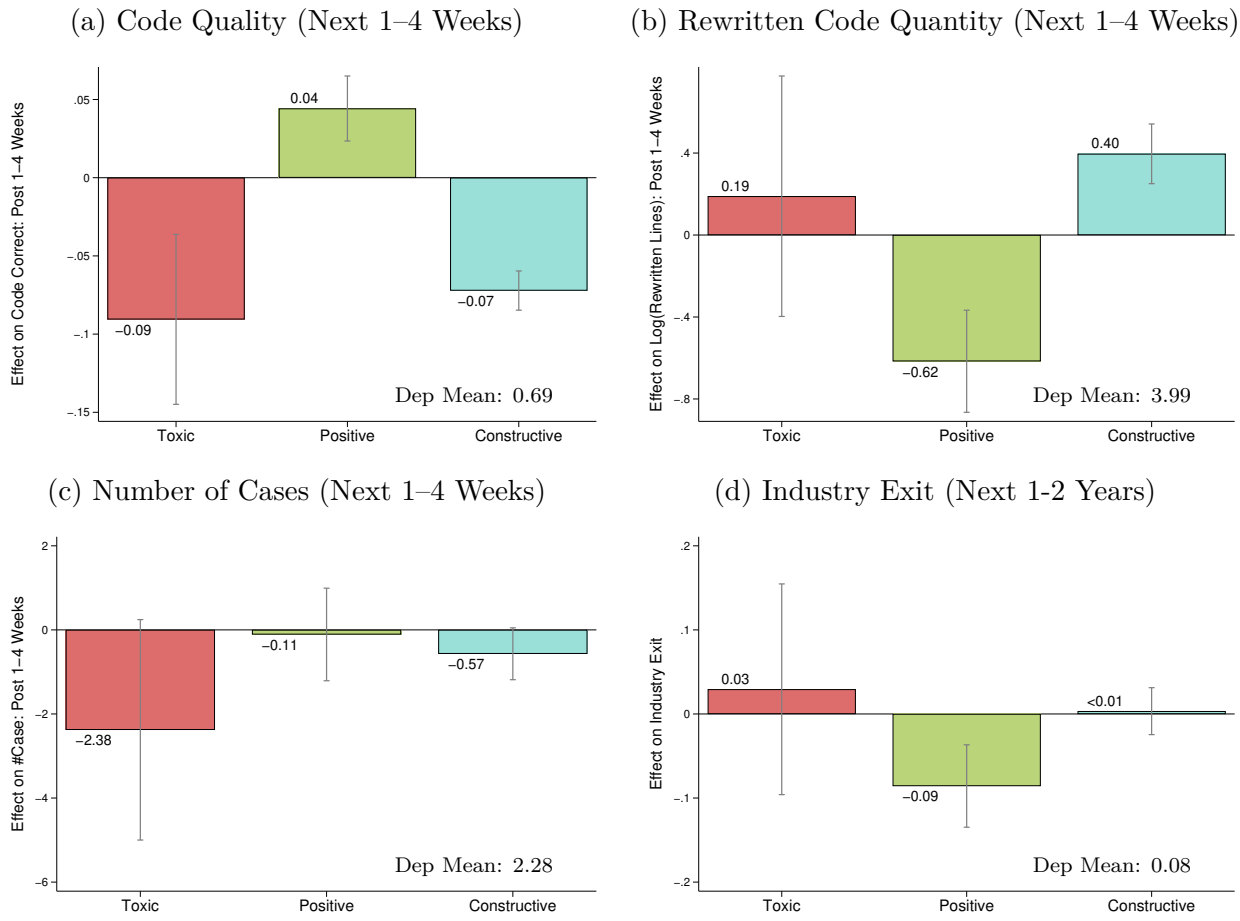


(d) Constructive Feedback



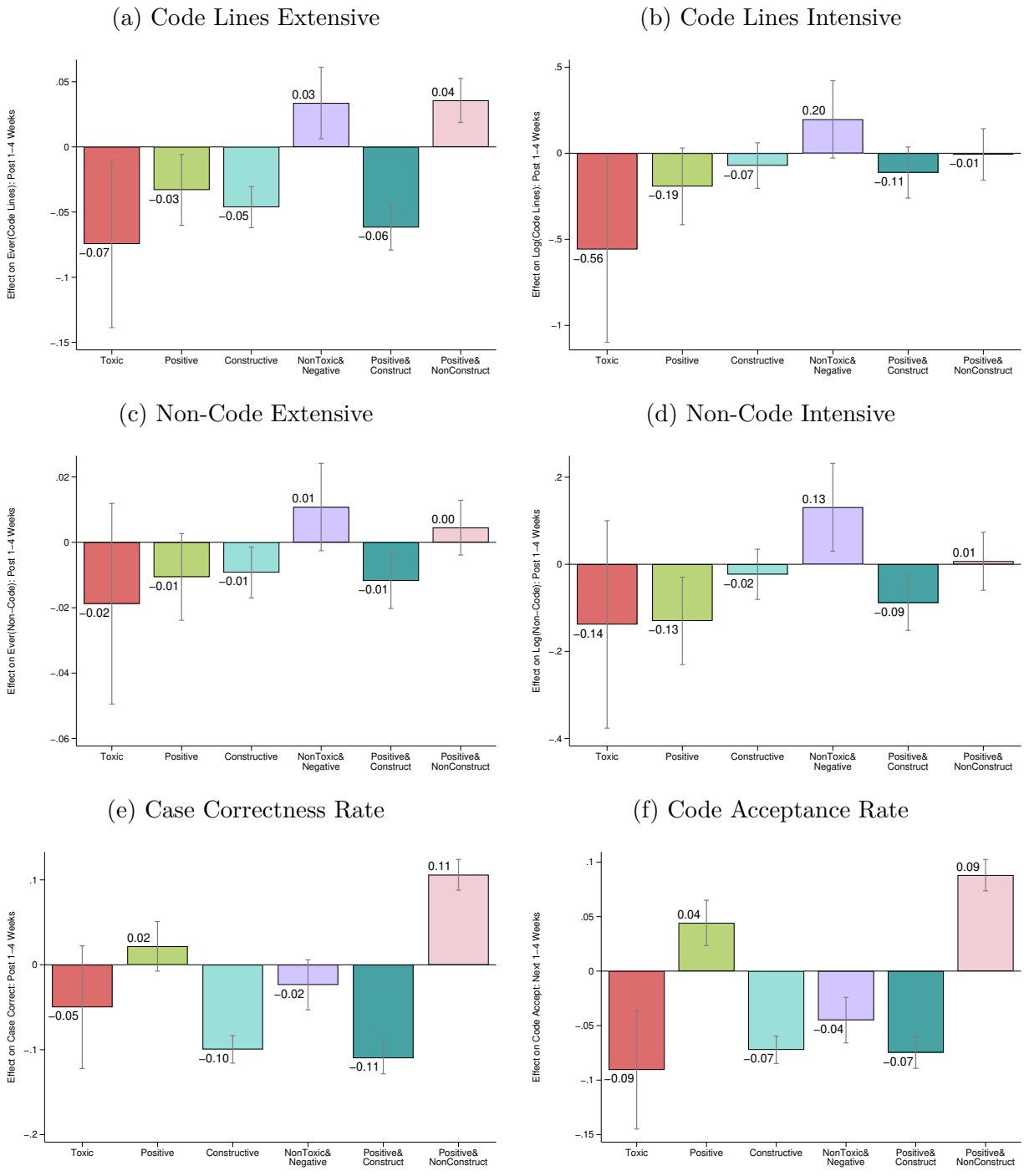
Notes: This figure reports two-stage least squares (2SLS) estimates of the effects of different feedback types on developers' subsequent productivity over one to four weeks: (a) toxic, (b) respectful criticism (negative and non-toxic), (c) positive, and (d) constructive. Productivity outcomes exclude the reviewed code, focusing on new code produced afterwards, and are measured at the developer–week level. *Code quantity* captures both the extensive margin (Code lines Ever) and the intensive margin (Code lines Log). *Non-code activity* covers GitHub tasks other than coding. *Code quality* measures the share of submissions merged into the team's main codebase (case correct) and the share of lines of code accepted after review (code accept). The regression instrument for feedback exposure using reviewer-level feedback tendencies within the random assignment sample. Lines denote 90% confidence intervals, and standard errors are clustered at the reviewer level.

Figure 9: Effect of Feedback on Developer Outcomes Within a Team



Notes: This figure reports 2SLS estimates of the effect of a given feedback type on developer outcomes within the same team during weeks 1–4 after the feedback. The sample includes teams with random reviewer assignment, at least three reviewers per year, and reviewers with at least 10 cases per year. The industry-exit and firm-switch outcomes come from LinkedIn; for these two outcomes, estimates use a 10% random subsample of the random-reviewer sample used in panels (a)–(d). Standard errors are clustered by reviewer, and lines show 90% confidence intervals.

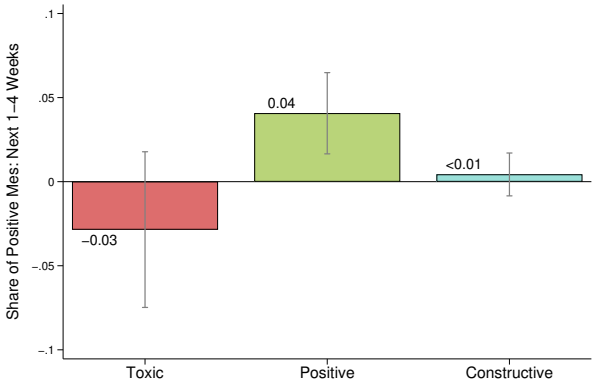
Figure 10: Effect of Feedback Interactions on Developer Productivity (Next 1–4 Weeks)



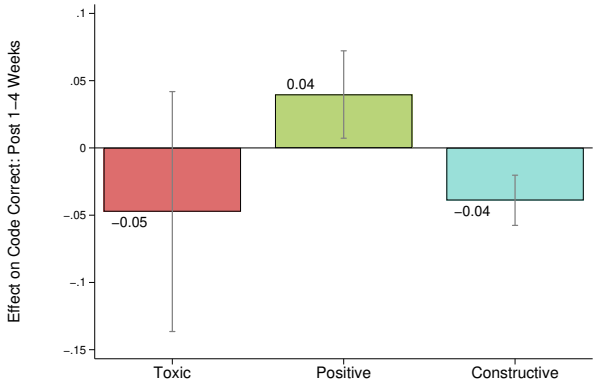
Notes: This figure reports two-stage least squares (2SLS) estimates of how different interactions of reviewer feedback affect developer productivity and code quality within a team over the following one to four weeks. Panels (a)–(d) show effects on code and non-code output, decomposed into extensive and intensive margins. Panels (e)–(f) report effects on code quality, measured by the case correctness rate and code acceptance rate. Lines denote 90% confidence intervals, and standard errors are clustered at the reviewer level.

Figure 11: Spillover Effects of Feedback (Next 1–4 Weeks)

(a) Share of Positive Messages to Coworkers

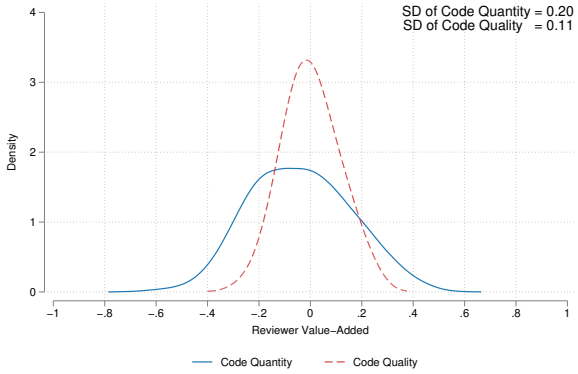


(b) Code Acceptance Rate in Other Teams

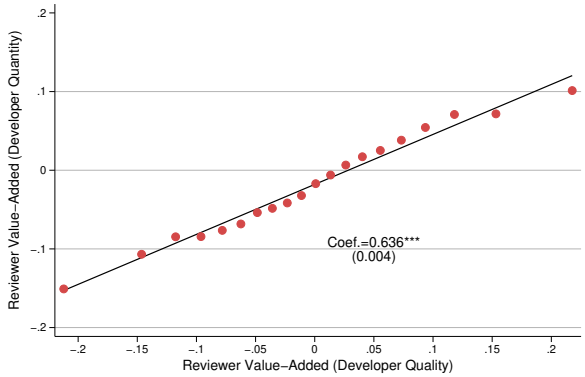


Notes: Panels (a) and (b) report two-stage least squares (2SLS) estimates of spillover effects of feedback over the following one to four weeks. **Panel (a)** shows how receiving feedback in the focal team affects the share of positive messages that developers send to their coworkers. **Panel (b)** reports effects on code acceptance rates in other teams. Lines denote 90% confidence intervals, and standard errors are clustered at the reviewer level.

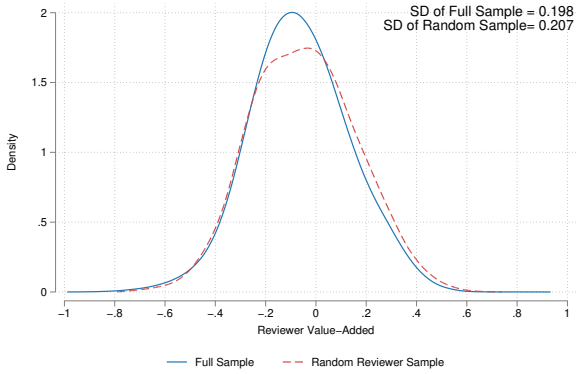
Figure 12: Reviewer Value-Added: Distributions and Correlation



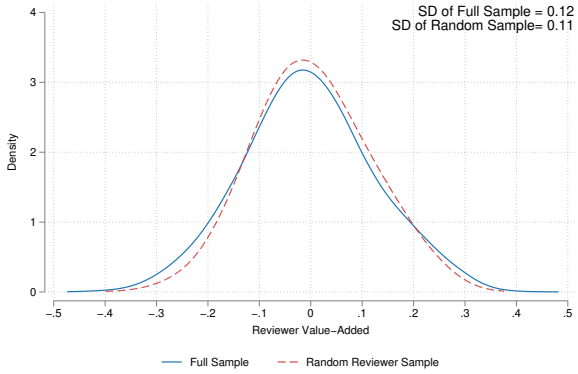
(a) Distribution of VA estimates



(b) Correlation of VA across output measures



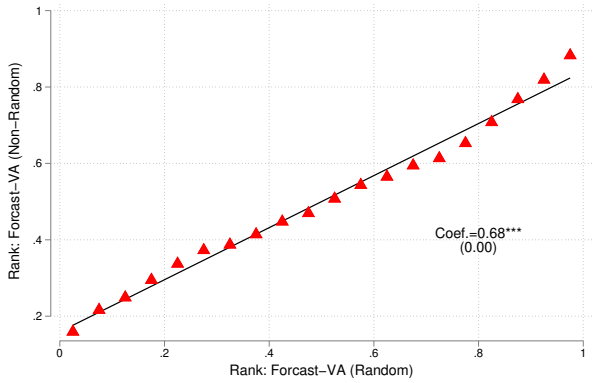
(c) VA based on code quantity



(d) VA based on code quality

Notes: Panels (a) and (b) present reviewer VA estimates using the random assignment sample. **Panel (a)** plots the empirical distribution of reviewer VA across productivity measures, and **Panel (b)** shows the correlation between VA based on code quantity and code quality. **Panels (c) and (d)** compare the distributions of reviewer VA measured in the random and full samples, using code quantity and code quality, respectively.

Figure 13: Reviewer Value-Added: Validation

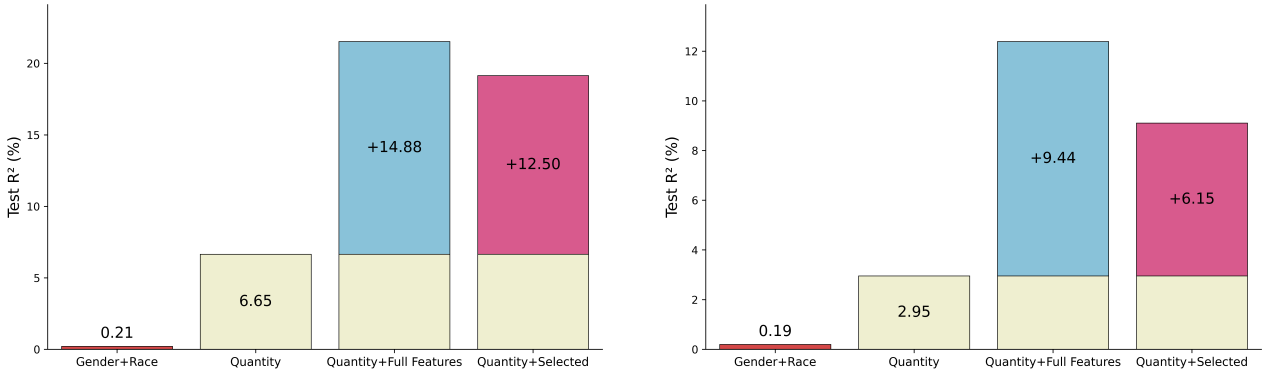


Notes: This figure presents a binscatter plot comparing reviewer VA rankings in both random and non-random samples. Each dot represents a bin of reviewers, and the slope and standard error are obtained from a regression of one ranking on the other. The sample includes only reviewers who appear in both the random and non-random assignment samples. There are 6,952 reviewers.

Figure 14: Feedback Characteristics Explain Variation in Reviewer Value-Added

(a) Random Sample

(b) Full Sample



Notes: This figure shows how different aspects of reviewer feedback explain the variation in reviewer value-added (VA), measured by its impact on developer code quantity. Panel (a) uses the random assignment sample; Panel (b) uses the full sample. Each bar reports the test R^2 from models using different inputs: *Quantity* is the number of feedback messages written; *Quantity+Embeddings* adds 768 semantic dimensions of the text; *Quantity+Features* includes the weekly shares of toxic, positive, and constructive feedback per reviewer. Values on the bars indicate the improvement in explanatory power relative to the quantity-only model. Figure G1 reports analogous results using developer code quality as the outcome.

Table 1: Descriptive Statistics

Description	Full Sample	Random Reviewer Sample
Panel A		
Team (#)	1,774,184	3,457
Case (#)	56,637,966	4,228,687
Case with reviewers (%)	17.11	33.42
Reviewers (#)	364,889	35,072
Avg. case per team (#)	21.92	1223.22
Avg. case per reviewer-team-week (#)	1.51	1.80
Avg. team size (#)	1.03	19.42
Avg. team create year	2020	2020
Panel B		
Message (#)	231,293,881	24,453,479
Avg. message per case (#)	4.08	5.78
Avg. message per team (#)	130.37	7073.61
Panel C: Reviewer Feedback (conditional on case with reviewers)		
Message sent by reviewers (%)	36.33	32.60
Toxic (% of reviewer feedback)	0.21	0.21
Positive (% of reviewer feedback)	41.87	39.40
Constructive (% of reviewer feedback)	34.65	31.36
Panel D		
Reviewers are junior to developers (%)	28.04	31.97
Reviewers who also write code within a team-year (%)	41.43	45.91
P(B reviews A | A reviews B) (%)	52.17	23.67
P(A is B's next reviewer | A reviews B) (%)	10.91	7.32

Notes: This table reports summary statistics for all teams (Full Sample) and for teams that adopted reviewer random assignment (Random Reviewer Sample) between 2017 and 2023. Team size is measured as the average number of members from 2017 to 2023. Team creation year refers to the first year a team appears in our data. **Panel B** includes messages sent by both team members and external users. **Panel C** focuses only on feedback messages sent by reviewers, which are the focus of this paper. Toxic and positive feedback are identified using a deep learning embedding model, while constructive feedback is labeled using an LLM. For the Random Reviewer Sample, all reviewer feedback messages are used. For the Full Sample, toxic and positive feedback are based on all reviewer feedback messages, while constructive feedback is based on a random subsample due to budget constraints.

Table 2: Developer-Team Weekly Productivity Summary Statistics

	N	Mean	Std. Dev	P50	P75	P95	Max
Focal Team							
Code Lines	12,216,340	1,119.21	54,898.64	0	3	537	41,436,660
Rewritten Lines	12,216,340	861.94	56,482.51	0	0	97	29,285,786
Non-Code Activities	12,216,340	6.36	39.89	2	5	26	62,784
Case	12,216,340	3.18	11.83	1	3	13	16,722
Case Correctness Rate	3,607,865	0.63	0.42	1	1	1	1
Code Acceptance Rate	4,095,714	0.74	0.38	1	1	1	1
Other Teams							
Code Lines	12,216,340	2,812.08	199,277.21	0	0	617	78,347,840
Rewritten Lines	12,216,340	861.94	56,482.51	0	0	97	29,285,786
Non-Code Activities	12,216,340	549.94	2,938.83	10	43	1,819	62,801
Case	12,216,340	85.41	457.56	2	16	97	16,741
Case Correctness Rate	2,054,565	0.67	0.34	1	1	1	1
Code Acceptance Rate	2,467,433	0.70	0.40	1	1	1	1

Notes: This table shows the summary statistics of productivity measures for all developers in the random reviewer sample from 2017 to 2023. Each observation is a developer-team-week. "Focal Team" refers to the team in the current row; "Other Teams" aggregates activity from all remaining teams during the same week. Winsorization at the 95th percentile is applied to all continuous variables to reduce the influence of extreme outliers, which are common in developer productivity data (e.g., unusually large code commits or bursts of activity). Case correctness rate and code acceptance rate are unaffected by winsorization. See Appendix A.4 and Appendix A.5 for variable definitions. Reported standard deviations reflect the original (pre-winsorization) distributions.

Table 3: Share of Feedback and Examples by Toxicity, Positivity, and Constructiveness

Panel A: Share of Feedback Types

Constructive

	Positive	**Negative**
Toxic	0.000	0.01
Non-Toxic	9.43	21.91

Non-Constructive

	Positive	**Negative**
Toxic	0.02	0.17
Non-Toxic	29.95	38.50

Panel B: Examples of Feedback Types

Toxic	**Positive**	**Constructive**	**Feedback Message**	**Share (%)**
1	1	0	Sounds good, can't wait to restaple my ass on.	0.02
1	0	1	You should just keep the file name, not the full file path. This is Pantherinternal and doesn't provide value...	0.01
1	0	0	You will have to answer for it and you are talking nonsense.	0.17
0	1	1	Docs look good overall. I think it would also be helpful to add a few more screenshots showing the overall page...	9.43
0	1	0	Looks good to me.	29.95
0	0	1	Still feels redundant. An angle should just be a union of float and FreeParameter. Once the parameter is bound...	21.91
0	0	0	We need a way to detect the cluster domain.	38.50

Notes: **Panel A** report classification results for reviewer feedback in the random reviewer sample, cross-classified by toxicity, positivity, and constructiveness. The entries across all cells sum to 100. **Panel B** gives example messages for each category. "Share" corresponds to the same sample as in **Panel A**.

Table 5: First-Stage Estimates by Feedback Type

	(1)	(2)	(3)
Panel A: Toxic Feedback			
Reviewer Toxicity	0.708***	0.703***	0.703***
	(0.013)	(0.013)	(0.013)
F-Stat.	2931.913	2866.244	2868.936
Dependent mean	0.008	0.008	0.008
Panel B: Positive Feedback			
Reviewer Positivity	0.843***	0.820***	0.820***
	(0.005)	(0.005)	(0.005)
F-Stat.	34335.891	28057.043	27450.902
Dependent mean	0.706	0.706	0.706
Panel C: Constructive Feedback			
Reviewer Constructiveness	0.875***	0.864***	0.863***
	(0.004)	(0.004)	(0.004)
F-Stat.	61420.629	60183.796	60733.265
Dependent mean	0.518	0.518	0.518
Team FE	✓		
Team × Year FE		✓	✓
Baseline controls			✓
N	1,179,525	1,179,525	1,179,525

Notes: Each panel reports a first-stage regression where the dependent variable is an indicator for receiving a specific type of feedback. Baseline controls include developer gender, race, GitHub activity in the year before submission, and the first year of GitHub activity. Case characteristics account for issue labels such as Bug and Enhancement, and project stage is defined based on the release version of the team product. Appendix A.1 provides details on the gender and race variables, and Appendix A.6 describes the construction of project stage. Standard errors are clustered at the reviewer level. *$p < .10$, ** $p < .05$, *** $p < .01$.

Table 6: Effect of Reviewer Feedback on Developer Productivity Within a Team

	Code Quantity		**Non-Code Quantity**		**Code Quality**	
	Ever	Log	Ever	Log	Case	Code
Panel A: Toxic Feedback						
OLS	-0.013***	0.065***	0.004***	0.080***	-0.007***	0.001
	(0.003)	(0.015)	(0.001)	(0.008)	(0.002)	(0.002)
RF	-0.057*	-0.421*	-0.014	-0.106	-0.038	-0.068***
	(0.030)	(0.250)	(0.014)	(0.111)	(0.033)	(0.025)
2SLS	-0.074*	-0.558*	-0.019	-0.138	-0.050	-0.091***
	(0.039)	(0.328)	(0.019)	(0.145)	(0.044)	(0.033)
Panel B: Positive Feedback						
OLS	0.015***	0.178***	0.010***	0.137***	-0.005***	-0.003**
	(0.002)	(0.008)	(0.001)	(0.004)	(0.001)	(0.001)
RF	-0.009**	-0.050	-0.003	-0.035**	0.006	0.012***
	(0.005)	(0.035)	(0.002)	(0.016)	(0.005)	(0.003)
2SLS	-0.033**	-0.192	-0.011	-0.130**	0.022	0.044***
	(0.017)	(0.136)	(0.008)	(0.061)	(0.018)	(0.013)
Panel C: Constructive Feedback						
OLS	0.017***	0.184***	0.009***	0.122***	-0.019***	-0.015***
	(0.001)	(0.007)	(0.001)	(0.003)	(0.001)	(0.001)
RF	–0.017***	–0.025	–0.003*	–0.008	-0.035***	-0.026***
	(0.004)	(0.028)	(0.002)	(0.013)	(0.003)	(0.003)
2SLS	–0.046***	–0.072	–0.009**	–0.023	-0.100***	–0.072***
	(0.010)	(0.080)	(0.005)	(0.035)	(0.010)	(0.008)
Dependent mean	0.846	5.018	0.957	2.450	0.545	0.693
N	1,176,459	995,139	1,176,459	1,125,874	998,141	1,028,793
Team × Year FE	✓	✓	✓	✓	✓	✓
Decile d–1 activity	✓	✓	✓	✓	✓	✓
Decile d–1 message	✓	✓	✓	✓	✓	✓

Notes: This table presents OLS, reduced form (RF), and two-stage least squares (IV) estimates of the effect of reviewer feedback. All statistics reported are for the sample used in IV. Panels A, B, and C correspond to toxic, positive, and constructive reviewer feedback, respectively. The outcome variables are developer productivity measures within the team during weeks 1–4 after receiving feedback. All models include team-by-year fixed effects. Standard errors are clustered at the reviewer level. * $p < .10$, ** $p < .05$, *** $p < .01$

References

Abowd, J. M., F. Kramarz, and D. N. Margolis (1999). High wage workers and high wage firms. *Econometrica 67*(2), 251–333.

Adams-Prassl, A., K. Huttunen, E. Nix, and N. Zhang (2024). Violence against women at work. *The Quarterly Journal of Economics 139*(2), 937–991.

Adhvaryu, A., A. Nyshadham, and J. Tamayo (2023). Managerial quality and productivity dynamics. *The Review of Economic Studies 90*(4), 1569–1607.

Agan, A., J. L. Doleac, and A. Harvey (2023). Misdemeanor prosecution. *The Quarterly Journal of Economics 138*(3), 1453–1505.

Andersen-Gott, M., G. Ghinea, and B. Bygstad (2012). Why do commercial companies contribute to open source software? *International Journal of Information Management 32*(2), 106–117.

Andreoni, J., W. Harbaugh, and L. Vesterlund (2003). The carrot or the stick: Rewards, punishments, and cooperation. *American Economic Review 93*(3), 893–902.

Andrews, M. J., L. Gill, T. Schank, and R. Upward (2008). High wage workers and low wage firms: negative assortative matching or limited mobility bias? *Journal of the Royal Statistical Society Series A: Statistics in Society 171*(3), 673–697.

Angrist, J. and G. Imbens (1994). Identification and estimation of local average treatment effects.

Angrist, J. D. and B. Frandsen (2022). Machine labor. *Journal of Labor Economics 40*(S1), S97–S140.

Ashford, S. J. and L. L. Cummings (1983). Feedback as an individual resource: Personal strategies of creating information. *Organizational behavior and human performance 32*(3), 370–398.

Bacher-Hicks, A. and C. Koedel (2023). Estimation and interpretation of teacher value added in research applications. *Handbook of the Economics of Education 6*, 93–134.

Bandiera, O., I. Barankay, and I. Rasul (2007). Incentives for managers and inequality among workers: Evidence from a firm-level experiment. *The Quarterly Journal of Economics 122*(2), 729–773.

Bandiera, O., A. Prat, S. Hansen, and R. Sadun (2020). Ceo behavior and firm performance. *Journal of Political Economy 128*(4), 1325–1369.

Banihashem, S. K., O. Noroozi, S. Van Ginkel, L. P. Macfadyen, and H. J. Biemans (2022). A systematic review of the role of learning analytics in enhancing feedback practices in higher education. *Educational Research Review 37*, 100489.

Becker, G. S. (1966). Human capital: A theoretical and empirical analysis, with special reference to education the residual factor and economic growth econometric models of education.

Bekker, P. A. (1994). Alternative approximations to the distributions of instrumental variable estimators. *Econometrica: Journal of the Econometric Society*, 657–681.

Bénabou, R. and J. Tirole (2002). Self-confidence and personal motivation. *The Quarterly Journal of Economics 117*(3), 871–915.

Bender, S., N. Bloom, D. Card, J. Van Reenen, and S. Wolter (2018). Management practices, workforce selection, and productivity. *Journal of Labor Economics 36*(S1), S371–S409.

Bertrand, M. and A. Schoar (2003). Managing with style: The effect of managers on firm policies. *The Quarterly Journal of Economics 118*(4), 1169–1208.

Beuschlein, J. (2024). Designing debt restructuring: The adverse effects on labor market outcomes. Technical report, Working Paper. Stockholm University.

Bhuller, M., G. B. Dahl, K. V. Løken, and M. Mogstad (2020). Incarceration, recidivism, and employment. *Journal of Political Economy 128*(4), 1269–1324.

Bloom, N., B. Eifert, A. Mahajan, D. McKenzie, and J. Roberts (2013). Does management matter? evidence from india. *The Quarterly Journal of Economics 128*(1), 1–51.

Bloom, N. and J. Van Reenen (2007). Measuring and explaining management practices across firms and countries. *The Quarterly Journal of Economics 122*(4), 1351–1408.

Bonhomme, S., T. Lamadon, and E. Manresa (2019). A distributional framework for matched employer employee data. *Econometrica 87*(3), 699–739.

Brynjolfsson, E., D. Li, and L. Raymond (2025). Generative ai at work. *The Quarterly Journal of Economics*, qjae044.

Card, D., A. Mas, E. Moretti, and E. Saez (2012). Inequality at work: The effect of peer salaries on job satisfaction. *American Economic Review 102*(6), 2981–3003.

Chetty, R., J. N. Friedman, and J. E. Rockoff (2014). Measuring the impacts of teachers i: Evaluating bias in teacher value-added estimates. *American Economic Review 104*(9), 2593–2632.

Collis, M. R. and C. Van Effenterre (2025). Workplace hostility. Technical report, Working Paper.

Compte, O. and A. Postlewaite (2004). Confidence-enhanced performance. *American Economic Review 94*(5), 1536–1557.

Crawford, V. P. and J. Sobel (1982). Strategic information transmission. *Econometrica: Journal of the Econometric Society*, 1431–1451.

Diaz, B. S., A. N. Nazarrett, J. Ramirez, R. Sadun, and J. A. Tamayo (2025). Training within firms. Technical report, National Bureau of Economic Research.

Dobbie, W., J. Goldin, and C. S. Yang (2018). The effects of pre-trial detention on conviction, future crime, and employment: Evidence from randomly assigned judges. *American Economic Review 108*(2), 201–240.

Dobbie, W., P. Goldsmith-Pinkham, and C. S. Yang (2017). Consumer bankruptcy and financial health. *Review of Economics and Statistics 99*(5), 853–869.

Dobbie, W. and J. Song (2015). Debt relief and debtor outcomes: Measuring the effects of consumer bankruptcy protection. *American Economic Review 105*(3), 1272–1311.

Dube, A., L. Giuliano, and J. Leonard (2019). Fairness and frictions: The impact of unequal raises on quit behavior. *American Economic Review 109*(2), 620–663.

Ederer, F., P. Goldsmith-Pinkham, and K. Jensen (2024). Anonymity and identity online. *arXiv preprint arXiv:2409.15948*.

El-Komboz, L. A. and M. Goldbeck (2024). Career concerns as public good the role of signaling for open source software development. Technical report, Ifo Working Paper.

Emanuel, N., E. Harrington, and A. Pallais (2025). The power of proximity to coworkers. Technical report, National Bureau of Economic Research.

Fattori, A., L. Neri, E. Aguglia, A. Bellomo, A. Bisogno, D. Camerino, B. Carpiniello, A. Cassin, G. Costa, P. De Fazio, et al. (2015). Estimating the impact of workplace bullying: humanistic and economic burden among workers with chronic medical conditions. *BioMed research international 2015*(1), 708908.

Folke, O. and J. Rickne (2022). Sexual harassment and gender inequality in the labor market. *The Quarterly Journal of Economics 137*(4), 2163–2212.

Frandsen, B., L. Lefgren, and E. Leslie (2023). Judging judge fixed effects. *American Economic Review 113*(1), 253–277.

Frederiksen, A., L. B. Kahn, and F. Lange (2020). Supervisors and performance management systems. *Journal of Political Economy 128*(6), 2123–2187.

Freimane, M. (2024). Gender bias, feedback, and productivity.

Gabaix, X. and A. Landier (2008). Why has ceo pay increased so much? *The quarterly journal of economics 123*(1), 49–100.

Gallup (2022). How effective feedback fuels performance. Retrieved April 19, 2025, from https://www.gallup.com/workplace/357764/fast-feedback-fuels-performance.aspx.

Gelfand, M. J., J. L. Raver, L. Nishii, L. M. Leslie, J. Lun, B. C. Lim, L. Duan, A. Almaliach, S. Ang, J. Arnadottir, et al. (2011). Differences between tight and loose cultures: A 33-nation study. *science 332*(6033), 1100–1104.

Gibbons, R. and M. Waldman (1999). A theory of wage and promotion dynamics inside firms. *The Quarterly Journal of Economics 114*(4), 1321–1358.

Gibbons, R. and M. Waldman (2004). Task-specific human capital. *American Economic Review 94*(2), 203–207.

GitHub (2025). Github enterprise. Accessed: August 27, 2025.

Haegele, I. (2022). Talent hoarding in organizations. *arXiv preprint arXiv:2206.15098*.

Hanushek, E. (1971). Teacher characteristics and gains in student achievement: Estimation using micro data. *American Economic Review 61*(2), 280–288.

Hartmann, J., M. Heitmann, C. Siebert, and C. Schamp (2023). More than a feeling: Accuracy and application of sentiment analysis. *International Journal of Research in Marketing 40*(1), 75–87.

Hartvigsen, T., S. Gabriel, H. Palangi, M. Sap, D. Ray, and E. Kamar (2022). Toxigen: A large-scale machine-generated dataset for implicit and adversarial hate speech detection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

Hattie, J. and H. Timperley (2007). The power of feedback. *Review of educational research 77*(1), 81–112.

Heckman, J. J. and E. Vytlacil (2005). Structural equations, treatment effects, and econometric policy evaluation 1. *Econometrica 73*(3), 669–738.

Hoffman, M. and S. Tadelis (2021). People management skills, employee attrition, and manager rewards: An empirical analysis. *Journal of Political Economy 129*(1), 243–285.

Hoffmann, M., F. Nagle, and Y. Zhou (2024). The value of open source software. *Harvard Business School Strategy Unit Working Paper* (24-038).

Holmström, B. (1979). Moral hazard and observability. *The Bell journal of economics*, 74–91.

Holmstrom, B. (1982). Moral hazard in teams. *The Bell Journal of Economics*, 324–340.

Hull, P. (2017). Examiner designs and first-stage f statistics: A caution. *Brown University Working Paper 5*.

Ichniowski, C., K. L. Shaw, and G. Prennushi (1997). The effects of human resource management practices on productivity.

Jones, B. F. (2021). The rise of research teams: Benefits and costs in economics. *Journal of Economic Perspectives 35*(2), 191–216.

Kahneman, D. (1973). *Attention and effort*, Volume 1063. Citeseer.

Kane, T. J. and D. O. Staiger (2008). Estimating teacher impacts on student achievement: An experimental evaluation. Technical report, National Bureau of Economic Research.

Kim, S. Y., Y. Wang, D. Orozco-Lapray, Y. Shen, and M. Murtuza (2013). Does "tiger parenting" exist? parenting profiles of chinese americans and adolescent developmental outcomes. *Asian American Journal of Psychology 4*(1), 7.

Kluger, A. N. and A. DeNisi (1996). The effects of feedback interventions on performance: a historical review, a meta-analysis, and a preliminary feedback intervention theory. *Psychological Bulletin 119*(2), 254.

Kolesár, M. (2013). Estimation in an instrumental variables model with treatment effect heterogeneity. Technical report.

Kolesár, M., R. Chetty, J. Friedman, E. Glaeser, and G. W. Imbens (2015). Identification and inference with many invalid instruments. *Journal of Business & Economic Statistics 33*(4), 474–484.

Kolhatkar, V., N. Thain, J. Sorensen, L. Dixon, and M. Taboada (2020). Classifying constructive comments. *arXiv preprint arXiv:2004.05476*.

Kőszegi, B., G. Loewenstein, and T. Murooka (2022). Fragile self-esteem. *The Review of Economic Studies 89*(4), 2026–2060.

Laohaprapanon, S., G. Sood, and B. Naji (2017). Ethnicolr algorithm.

Lazear, E. P. (2000). Performance pay and productivity. *American Economic Review 90*(5), 1346–1361.

Lazear, E. P., K. L. Shaw, and C. T. Stanton (2015). The value of bosses. *Journal of Labor Economics 33*(4), 823–861.

Lee, D. S., J. McCrary, M. J. Moreira, and J. Porter (2022). Valid t-ratio inference for iv. *American Economic Review 112*(10), 3260–3290.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

London, M. (2003). *Job feedback: Giving, seeking, and using feedback for performance improvement*. Psychology Press.

Lucas Jr, R. E. (1978). On the size distribution of business firms. *The Bell Journal of Economics*, 508–523.

Marschak, J. and R. Radner (1958). Economic theory of teams. chapter 5.

McIntosh, S., Y. Kamei, B. Adams, and A. E. Hassan (2016). An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering 21*, 2146–2189.

McKinsey (2024). Feedback culture: Great learning design as a bridge to culture building. Accessed: August 27, 2025.

McTernan, W. P., M. F. Dollard, and A. D. LaMontagne (2013). Depression in the workplace: An economic cost analysis of depression-related productivity loss attributable to job strain and bullying. *Work & Stress 27*(4), 321–338.

Metcalfe, R. D., A. B. Sollaci, and C. Syverson (2023). Managers and productivity in retail. Technical report, National Bureau of Economic Research.

Minni, V. M. L. (2024). Making the invisible hand visible: Managers and the allocation of workers to jobs.

Mirrlees, J. (1974). Notes on welfare economics, information and uncertainty. *Essays on economic behavior under uncertainty*, 243–261.

Mirrlees, J. A. (1976). The optimal structure of incentives and authority within an organization. *The Bell Journal of Economics*, 105–131.

Nagle, F. (2019). Open source software and firm productivity. *Management Science 65*(3), 1191–1215.

Nekoei, A., J. Sigurdsson, and D. Wehr (2024). The economic burden of burnout.

Nielsen, M. B. and S. Einarsen (2012). Outcomes of exposure to workplace bullying: A meta-analytic review. *Work & Stress 26*(4), 309–332.

O*NET (2025). Management occupations. https://www.onetonline.org/find/family?f=11. Retrieved August 19, 2025.

Pinquart, M. and R. Kauser (2018). Do the associations of parenting styles with behavior problems and academic achievement vary by culture? results from a meta-analysis. *Cultural Diversity & Ethnic Minority Psychology 24*(1), 75.

Rothstein, J. (2010). Teacher quality in educational production: Tracking, decay, and student achievement. *The Quarterly Journal of Economics 125*(1), 175–214.

Rothstein, J. (2017). Measuring the impacts of teachers: Comment. *American Economic Review 107*(6), 1656–1684.

Sarker, J., A. K. Turzo, and A. Bosu (2020). A benchmark study of the contemporary toxicity detectors on software engineering interactions. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 218–227. IEEE.

Saygin, P., T. Knight, R. Ainsworth, D. Atchison, D. Card, D. Clark, C. Curran, M. Davia, M. Hoekstra, N. Iriberri, et al. (2023). *Gender Bias in Performance Evaluations: Evidence from a Field Experiment*. SSRN.

Shapiro, C. and J. E. Stiglitz (1984). Equilibrium unemployment as a worker discipline device. *The American economic review 74*(3), 433–444.

Shastry, G. K., O. Shurchkov, and L. L. Xia (2020). Luck or skill: How women and men react to noisy feedback. *Journal of Behavioral and Experimental Economics 88*, 101592.

Smirnov, I., C. Oprea, and M. Strohmaier (2023). Toxic comments are associated with reduced activity of volunteer editors on wikipedia. *PNAS nexus 2*(12), pgad385.

Solow, R. M. (1957). Technical change and the aggregate production function. *The review of Economics and Statistics 39*(3), 312–320.

Staiger, D. O. and J. H. Stock (1994). Instrumental variables regression with weak instruments.

Sutton, R. and B. Wigert (2019). More harm than good: The truth about performance reviews. *GALLUP: Workplace*.

U.S. Bureau of Economic Analysis (2025). Digital economy. https://www.bea.gov/data/special-topics/digital-economy. Retrieved September 8, 2025, from https://www.bea.gov/data/special-topics/digital-economy.

Vasilescu, B., Y. Yu, H. Wang, P. Devanbu, and V. Filkov (2015). Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 805–816.

Wagner, S. and M. Ruhe (2018). A systematic review of productivity factors in software development. *arXiv preprint arXiv:1801.06475*.

Weidmann, B., J. Vecci, F. Said, D. J. Deming, and S. R. Bhalotra (2025). How do you find a good manager? Technical report, National Bureau of Economic Research.

Zeltzer, D., L. Einav, A. Finkelstein, T. Shir, S. M. Stemmer, and R. D. Balicer (2023). Why is end-of-life spending so high? evidence from cancer patients. *Review of Economics and Statistics 105*(3), 511–527.

Appendix

A Demographics and Productivity Measures

A.1 Gender Prediction

To predict the gender of developers based on their profile page avatars, I select a deep-learning model that closely matches the images in the training sample in terms of resolution and cropping methods. This chosen model[27] trains vision transformers on male and female portraits and develops a corresponding image classifier. The model predicts the probability that an image represents a female or a male. To determine the optimal cutoff for gender classification, I use a sample labeled by some annotators and find that the highest accuracy is achieved with a cutoff of 0.85. If the predicted probability is above 0.85, the image will be classified as male or female; otherwise, it will be labeled as "missing". I then create a variable *Male*, which equals 1 if the image is classified as male, 0 if it is female, and 3 if the classification is missing.

A.2 Race Prediction

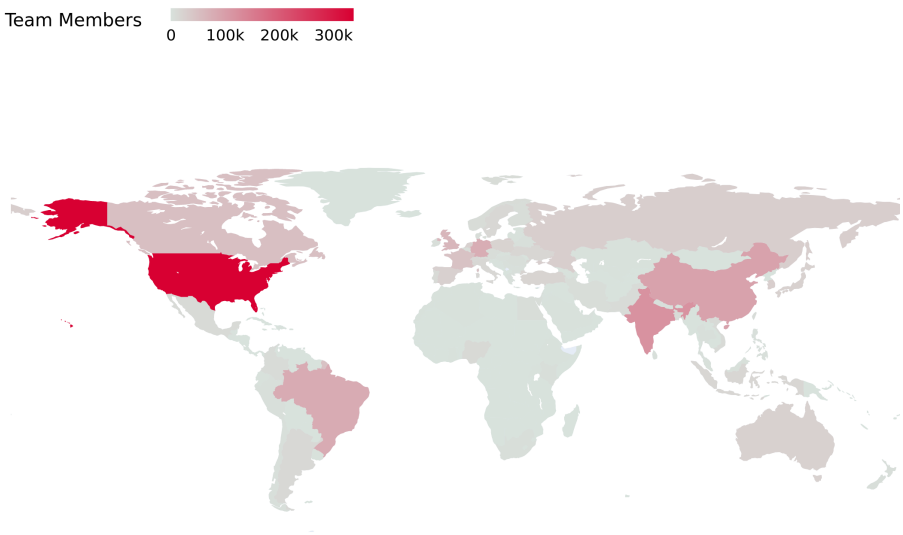
I use Ethnicolr, developed by Laohaprapanon et al. (2017), to predict developers' race and ethnicity from their names. The algorithm uses first and last names to classify individuals into three categories: Asian, Black, and White. It is trained on data from the US Census, Florida voter registration, and Wikipedia. For each name, Ethnicolr assigns probabilities for belonging to each category, and the category with the highest probability is taken as the prediction. In the heterogeneous analysis, I construct the variable *Asian*, which equals 1 if the predicted race is Asian, 0 if non-Asian, and 3 if missing. I define *White* analogously.

A.3 Location Prediction

I built the country variable in three steps. Between October 2023 and March 2024, I scraped each developer's free-text "location" field from their GitHub profile. I then standardized entries—e.g., "Québec" → "Canada," "北京" → "China," "São Paulo" → "Brazil", "München" → "Germany" . Finally, I assigned each user to exactly one country or flagged the entry as missing if it failed to match any country. The resulting variable records each developer's self-reported nation of residence.

[27] https://huggingface.co/rizvandwiki/gender-classification, accessed 2024-04-23

Figure A1: Geographic Distribution of Team Members



Notes: This world map shows the number of team members by country, based on their reported location. Team member is defined in Section 2.1.2. Darker red shades indicate higher team member counts. The United States has the largest concentration, followed by India, China, and Brazil. Participation is also notable in parts of Europe and Southeast Asia.

A.4 Code Quantity Measures

I distinguish between two types of code quantity based on the case (pull-request) lifecycle. Let t denote a week, i a developer, and k a case. I define the following weekly code-quantity measures for developer i:

New Code Lines

$$new\_code\_lines_{i,t} = \sum_{\substack{k: \\ \text{open\_week}(k)=t}} \text{additions}_{i,k,t,\text{opened}}$$

where $\text{additions}_{i,k,t,\text{opened}}$ is the number of lines added by developer i when submitting case k during week t. These new code lines capture the total lines first submitted by developer i in week t, reflecting new coding effort before any review.

For example, if in week 2020-01 developer i opens exactly one case k with 120 lines added,

$$new\_code\_lines_{i,\,\text{2020-01}} = 120.$$

Close Code Lines

$$close\_code\_quantity_{i,t} = \sum_{\substack{k: \\ \text{close\_week}(k)=t}} \text{additions}_{i,k,t,\text{closed}}$$

where $\text{additions}_{i,k,t,\text{closed}}$ is the number of lines added by developer i for a case k is closed during week t. This closed code quantity captures all lines merged into the codebase in week t after revision.

For example, if in week 2020-01 developer i merges exactly one case k with 200 lines added,

$$close\_code\_quantity_{i,\,\text{2020-01}} = 200.$$

Rewritten Code Lines

$$Rewritten\_code\_quantity_{i,t} = \sum_{\substack{k: \\ \text{close\_week}(k)=t}} \left(\left| \text{additions}_{i,k,t,\text{closed}} - \text{additions}_{i,k,\text{opened}} \right| \right)$$

Consider a developer i only works on one case $k = 10$ that opens in week $t =$ 2020-01 and closes in week $t + 1 =$ 2020-02. Suppose

$$\text{additions}_{i,10,\text{2020-01,opened}} = 120, \quad \text{additions}_{i,10,\text{2020-02,closed}} = 200.$$

Then:

$$new\_code\_lines_{i,\text{2020-01}} = 120, \quad new\_code\_lines_{i,\text{2020-02}} = 0,$$

and the rewritten code lines are

$$\left| \text{additions}_{i,10,\text{2020-02,closed}} - \text{additions}_{i,10,\text{opened}} \right| = |200 - 120| = 80.$$

A.5 Code Quality Measures

Case Correctness Rate

$$case\_correctness\_rate_{i,t} = \frac{\sum_k \mathbf{1}\{\text{correct}_k\}}{\sum_{\substack{k: \\ \text{open\_week}(k)=t}} 1},$$

where

$$\mathbf{1}\{\text{correct}_k\} = \begin{cases} 1, & \text{if the initial code submission for case } k \text{ is merged,} \\ 0, & \text{otherwise.} \end{cases}$$

$case\_correctness\_rate_{i,t}$ captures the share of developers' cases that meet review standards

without requiring substantial changes.

Code Acceptance Rate

$$code\_acceptance\_rate_{i,t} = \frac{new\_code\_lines_{i,t}}{total\_code\_quantity_{i,t}}.$$

This metric captures the proportion of a developer's submitted code in week t that required no further edits. A value of 1 indicates that every line added at case opening survived review unchanged, while lower values reflect more extensive post-submission revisions. Higher $code\_acceptance\_rate_{i,t}$ therefore signals better first-pass accuracy and overall code quality.

A.6 Project Stage

A GitHub *release* is the project's official, stable snapshot. Because each release package includes bug fixes, new features, or architectural changes, the sequence of releases offers a natural way to track a project's progress. Each release includes a *Semantic Versioning* (SemVer) tag in the format `MAJOR.MINOR.PATCH`, such as `v1.2.3`. The three components indicate the scale of change: the patch number increases for bug fixes, the minor number for backward-compatible feature additions, and the major number for breaking or incompatible changes.

Most modern projects adopt the SemVer tag, and the release tag provides a reliable and comparable signal of project stages. To classify project maturity, I extract the `MAJOR` and `MINOR` components from each release's Semantic Versioning tag. I then group releases into three stages. Stage 1 (Early) includes all tags with `MAJOR` $= 0$, as well as `1.0.0`, reflecting either pre-release development or an initial stable version. Stage 2 (Stable) covers releases with `MAJOR` $= 1$ and `MINOR` > 2, representing backward-compatible feature growth. Stage 3 (Mature) includes all releases with `MAJOR` ≥ 2, indicating that the project has undergone at least one major upgrade. [28] Projects that do not follow SemVer, for example, those using date stamps, commit hashes, or missing tags, are grouped into a separate "non-SemVer" category. This approach yields a simple, interpretable stage variable that can be compared across projects, much like how academic papers move through stages such as data collection, analysis, submission, and revision.

Assigning Cases to Project Stage. Let d_k denote the opening date of case k, and let $\{(s_i, t_i)\}_{i=1}^{N}$ represent the sequence of project releases, where s_i is the stage assigned to release

[28] Incrementing the `PATCH` number (e.g. 1.0.1 → 1.0.2) records a bug fix; raising the `MINOR` number (1.0.2 → 1.1.0) adds new, backward-compatible features; and advancing the `MAJOR` number (1.1.0 → 2.0.0) signals an intentional, incompatible redesign. See https://semver.org/ for more information.

i and t_i is the release date. The project stage at the time of case k is assigned as:

$$\text{stage}_k = s_{i^*} \quad \text{where} \quad i^* = \max\{i : t_i \leq d_k\}.$$

This procedure links each case to the most recently completed release stage available at the time the case was opened. Among cases in the random reviewer sample, 15.45% fall into Stage 1 (early development), 17.40% into Stage 2 (stable), 42.19% into Stage 3 (mature), and 24.96% are associated with a non-SemVer category.

B Random Reviewer Assignment

GH Archive does not record when teams enable random reviewer assignment, since such changes do not appear in the event timeline. Instead, I infer adoption by identifying the earliest pull request that modifies one of the configuration files associated with random assignment. Table B1 lists these files and explains their functions. By noting the first change to any of these files, I determine when each team activated the random reviewer assignment.

Table B1: Configuration Files Indicating Random Reviewer Assignment

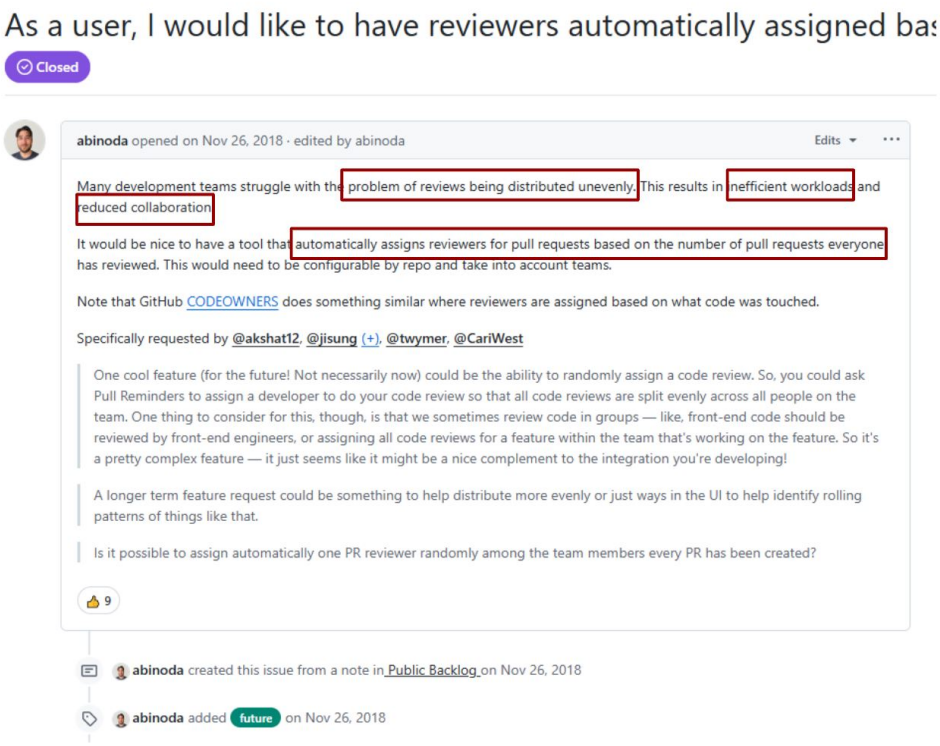
Configuration File	**Purpose**
`CODEOWNERS`	GitHub's native mechanism to assign reviewers by file path. Changes to matched files auto-request reviews from listed users or teams.
`.github/auto_assign.yml` or `.github/auto-assign.yml`	Config for the "auto-assign" Action: lists eligible reviewers and rules (e.g. random selection) for automatic assignment on pull request open.
`.github/reviewer_lottery.yml` or`.github/reviewer-lottery.yml`	Defines reviewer pools and lottery rules for custom bots that rotate or randomly pick reviewers per pull request.
`.github/assign_reviewers.yml` or `.github/assign-reviewers.yml`	Generic config used by GitHub Actions to specify reviewer pools and selection logic (e.g. round-robin, weighted).
`.github/workflows/assign_reviewers.yml` or `.github/workflows/assign-reviewers.yml`	A workflow that reads the above config and auto-assigns reviewers on pull request events.
`.github/workflows/auto_assign.yml` or `.github/workflows/auto-assign.yml`	Workflow file for an Action that implements automated reviewer selection (e.g. based on past workload or random choice).

Notes: This table lists the most common files and workflows used for random reviewer assignment. Additional custom or third-party configurations may also exist, but this selection reflects a conservative approach.

Figure B1 illustrates the motivation for random reviewer assignment, drawn from a discussion post highlighting the need to balance review workloads.

Figure B2 shows two scenarios: Figure B2a with automatic random assignment, and Figure B2b where the developer selects reviewers manually—either via "git blame" suggestions or

Figure B1: Reason for Automate Random Reviewer Assignment



Notes: The figure captures a discussion proposing automatic reviewer assignment using metrics such as each member's total pull requests reviewed. By allocating reviews based on these metrics, teams can achieve fairer workload distribution and strengthen collaboration. Original post, accessed 2025-01-31

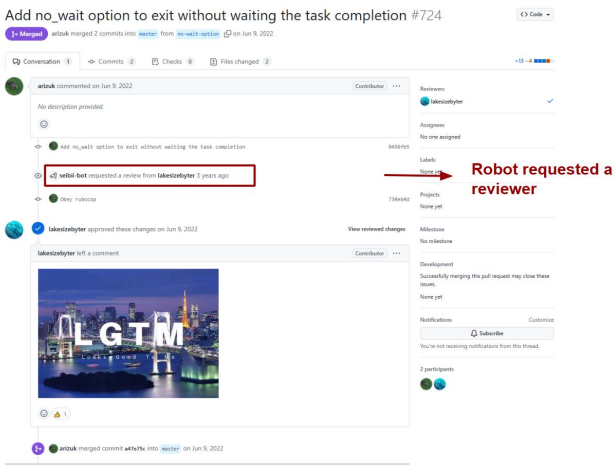
by typing usernames directly.

Figure B3 shows an example of the "Reviewer Roulette" random assignment workflow.

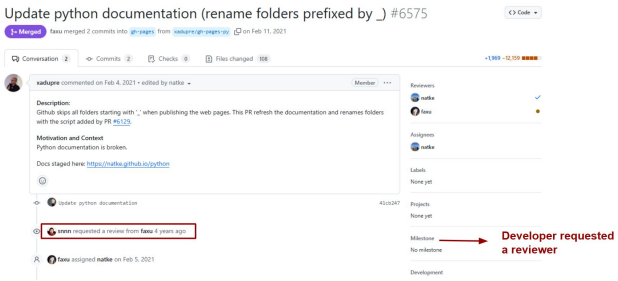
Table B2 reports the share and total number of teams in major tech firms adopting GitHub's random reviewer feature. Overall, 0.2% of teams use the feature (3,457 out of 1,744,184).

Figure B2: Random Reviewer Example

(a) Review requested by Robot

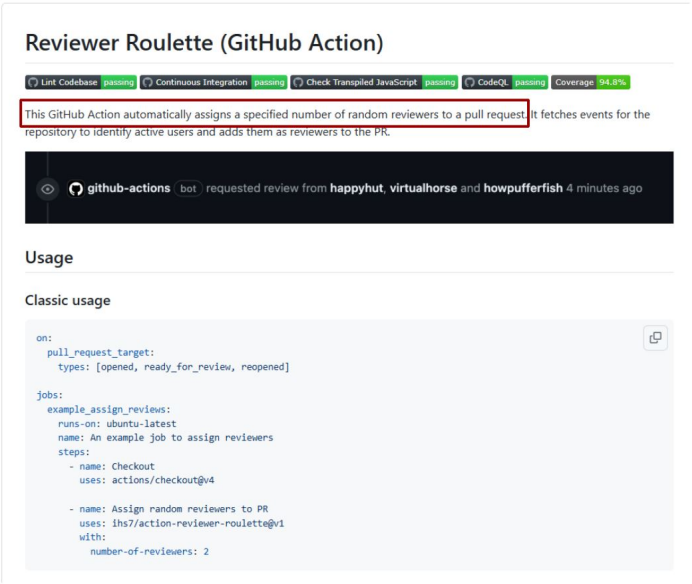


(b) Review requested by Developer



Notes: This graph illustrates two types of review requests. Figure B2a depicts a GitHub bot initiating a review request (random assignment), while Figure B2b shows a developer requesting a review.

Figure B3: GitHub Action for Random Reviewer Assignment



Notes: This figure illustrates the Reviewer Roulette GitHub Action, which automatically assigns a specified number of random reviewers to a pull request. The action fetches events from the repository to identify active users and assigns them as reviewers. The top section highlights the successful execution of this action, with random reviewers such as `happyhut`, `virtualhorse`, and `howpufferfish` being assigned. Detailed information from GitHub Marketplace Action, accessed 2025-01-31

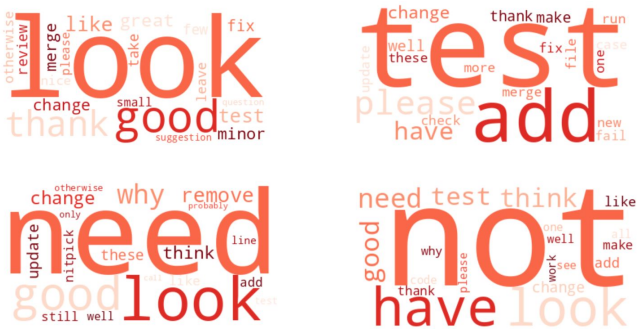
Table B2: Share of Teams in Big Tech Firms adopting Random Reviewer Feature

Firm	**Share(%)**	**Teams**	**Firm**	**Share(%)**	**Teams**
GitHub	3.226	341	Spotify	0.508	197
Pinterest	3.030	66	IBM	0.337	1780
Intel	1.173	597	Adobe	0.140	714
Apple	1.156	173	Uber	0.000	205
Netflix	1.156	173	PayPal	0.000	189
Salesforce	1.145	262	Yahoo	0.000	185
NVIDIA	1.124	267	Amazon	0.000	146
Microsoft	1.119	4645	Twitter	0.000	137
Stripe	0.917	109	Airbnb	0.000	120
Facebook	0.855	234	Dropbox	0.000	120
Oracle	0.800	250	eBay	0.000	100
OpenAI	0.794	126	Reddit	0.000	60
Google	0.723	1797	Zoom	0.000	43
Lyft	0.699	143	Snapchat	0.000	27
Tencent	0.585	171	Tesla	0.000	5

Notes: This table summarizes the share and total number of teams in major tech firms adopting the random reviewer feature. Overall, 0.2% of teams used the feature (3,516 out of 17,741,894). GitHub, Pinterest, and Intel show the highest adoption rates, with GitHub at 3.23% across 341 teams. Microsoft (4,645 teams) and Google (1,797 teams) also have above-average adoption. Firms like Uber, Twitter, and Tesla show no adoption during the sample period.

C Feedback Classification

Figure C1: Word Clouds of Principal Components Analysis (PCA) in Reviewer Feedback



Notes: This figure presents word clouds for the top four principal components extracted from reviewer feedback in the random reviewer sample. PCA reduces millions of feedback messages into a few components that capture most of the meaningful variation in the data. Each word cloud visualizes one component, where larger words appear more frequently within that dimension. The components consistently capture meaningful dimensions of feedback—such as positive (e.g., *good*, *thank*), negative (e.g., *not*), and constructive (e.g., *test*, *test*) language. A single word may load on multiple components if it conveys more than one meaning.

C.1 Toxicity

C.2 Constructiveness

I use large language models (LLMs) to detect constructive feedback on GitHub. I experimented with small changes in prompt wording, aiming to give the LLM a clear and objective instruction. I also tested including and excluding code snippets in the input. Neither change affected the model's output. I reviewed the prompt design with a software engineer from a major publicly listed technology firm and confirmed, using a small test set, that these variations do not alter the results.

My specific LLM prompt is as follows:

System Content:

You are an experienced principal software engineer reviewing a pull request comment. Rate its constructiveness on a scale from 0 (not constructive) to 10 (highly constructive). Focus on whether the comment provides specific or actionable information. A specific comment identifies a bug, suggests an improvement, or points to a concrete issue. If the comment is vague or lacks detail, treat that as a weakness. Focus on the content, not on the tone. Provide them in JSON format with the following key: info_score.

User Content:

Code: `<INPUT>`

Figure C2: Toxigen Annotation Setup for Evaluating

Notes: This is from Figure 6 of Hartvigsen et al. (2022), which shows how human annotators label toxicity for the Toxigen Classification Model.

Comment: `<INPUT>`

To identify the best-performing LLM for this task, I compare four OpenAI models (gpt-4.1-nano, gpt-4o-mini, gpt-4o, and gpt-o3) to find the one whose judgments most closely match those of software industry professionals, while minimizing cost. The initial human labels are produced by a master's student in computer science and verified by a software engineer.

The annotator rates each feedback item on a 0–10 scale: scores of 0–2 indicate little or no relevance or help, 3–5 indicate some relevance or help but with unclear specifics, and scores above 6 indicate strong relevance and high helpfulness, including detailed error explanations or suggestions that simplify code.

Once I obtain the ground truth and model predictions from different LLMs, I evaluate performance using binary rather than absolute scores. I evaluate performance using binary rather than absolute scores to reduce measurement error from minor discrepancies in human ratings. To generate binary labels, I test three decision cutoffs (≥ 3, ≥ 4, and ≥ 5). Each

cutoff changes the share of feedback classified as constructive and affects model performance. Table C1 reports results for each cutoff.

Table C1: Accuracy and F1 Scores by Model and Cutoff

Cutoff	gpt-4.1-nano		gpt-4o-mini		gpt-4o		gpt-o3	
	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
≥ 3	0.87	0.817	0.76	0.760	0.82	0.804	0.80	0.783
≥ 4	0.84	0.579	0.90	0.814	0.88	0.767	0.90	0.828
≥ 5	0.84	0.200	0.96	0.875	0.87	0.723	0.93	0.829

I select the gpt-4o-mini model because it offers the best balance between precision and recall, aligns closely with the human annotator's definition of clear and actionable feedback, and delivers performance comparable to higher-end models at a fraction of the cost. For example, it achieves an F1 score of 0.828 at a cost of about $0.93 to classify 10,000 feedback messages, compared with 0.829 and $23.25 for gpt-o3—a saving of more than 95%. I use a decision cutoff of 4 because it maintains a reasonable class balance (about 25% positives) and reflects the annotator's standard for clear and actionable feedback. I also report results for cutoff = 3 as a robustness check; model rankings remain consistent, indicating the findings are not sensitive to small threshold changes.

Table C2 presents examples of constructive feedback from the real data.

D Technical Appendix

D.1 Comparisons to Alternative Instrument Estimation Strategies

The main instrument is a residualized leave-out mean leniency score, based on a reviewer's feedback messages in cases involving other developers. Following standard practice (Dobbie and Song, 2015; Dobbie et al., 2017; Bhuller et al., 2020; Agan et al., 2023), I report F-statistics without correcting for the fact that the instrument is estimated. This approach has attractive properties in applied settings.

As Agan et al. (2023) notes, including reviewer fixed effects directly may lead to bias due to the large number of potentially weak instruments. In addition, my specification includes many fixed effects (e.g., team-by-year), which are necessary for identifying cases with as-if random reviewer assignment. Still, it may introduce bias in jackknife IV estimators (Kolesár, 2013). Using a continuous instrument also enables the estimation of marginal treatment effects (Heckman and Vytlacil, 2005).

While convenient, the main estimation strategy does not account for the fact that the instrument is itself estimated. To assess robustness, I explore several alternative approaches

Table C2: Examples of Reviewer Feedback and Constructiveness Scores

Feedback Message	**Constructive**	**Score**
Agree	0	0
Just checked one more time - now it's okay	0	1
Same advice on the assets point of view	0	2
Explain why there are private fields in this visitor. too multiple of them	0	3
I don't think all introspector errors are reported in the domain status. We need to make sure that all severe jrf ones are	1	4
You only need to define the acronym once, so I'd suggest doing it in the text not the title	1	5
This should be an async function that can use await and, as a result, standard trycatch blocks	1	6
We need to be generating a new nonce used to encrypt here rather than reusing the one that was used previously to encrypt the keyring. Otherwise there's a crypto bug in this that would allow an attacker to combine the old ciphertext with the new ciphertext in a way that could reveal the xor'd plaintext of the two messages.	1	10

commonly used when IV estimates may be biased due to many (potentially weak) instruments. Table D1 reports these results. Column (1) repeats the main 2SLS estimates using the residualized leave-out mean leniency score. Column (2) reports estimates from a limited information maximum likelihood (LIML) model using all reviewer dummies as instruments (Bekker, 1994; Angrist and Frandsen, 2022). Column (3) presents results from the modified bias-corrected two-stage least squares (MBTSLS) estimator of Kolesár et al. (2015), and column (4) shows estimates from the unbiased jackknife IV estimator (UJIVE) of Kolesár (2013).In each case, the estimated coefficients are nearly identical to the main 2SLS results.

Table D1: Different IV Estimation Strategies

	(1) Main	(2) LIML	(3) MBTSLS	(4) UJIVE
Panel A: Toxic Feedback				
Code Quantity Ever	-0.074*	-0.074*	-0.074*	-0.077**
	(0.039)	(0.039)	(0.039)	(0.039)
Code Quantity Log	-0.558*	-0.558*	-0.558*	-0.569*
	(0.328)	(0.328)	(0.328)	(0.328)
Non-Code Quantity Ever	-0.019	-0.019	-0.019	-0.020
	(0.019)	(0.019)	(0.019)	(0.019)
Non-Code Quantity Log	-0.138	-0.138	-0.138	-0.143
	(0.145)	(0.145)	(0.145)	(0.145)
Case Correct Rate	-0.050	-0.050	-0.050	-0.049
	(0.044)	(0.044)	(0.044)	(0.044)
Code Correct Rate	-0.091***	-0.091***	-0.091***	-0.091***
	(0.033)	(0.033)	(0.033)	(0.033)
Panel B: Positive Feedback				
Code Quantity Ever	-0.033**	-0.033**	-0.033**	-0.033**
	(0.017)	(0.017)	(0.017)	(0.017)
Code Quantity Log	-0.192	-0.192	-0.192	-0.190
	(0.136)	(0.136)	(0.136)	(0.135)
Non-Code Quantity Ever	-0.011	-0.011	-0.011	-0.011
	(0.008)	(0.008)	(0.008)	(0.008)
Non-Code Quantity Log	-0.130**	-0.130**	-0.130**	-0.130**
	(0.061)	(0.061)	(0.061)	(0.061)
Case Correct Rate	0.022	0.022	0.022	0.022
	(0.018)	(0.018)	(0.018)	(0.018)
Code Correct Rate	0.044***	0.044***	0.044***	0.044***
	(0.013)	(0.013)	(0.013)	(0.013)
Panel C: Constructive Feedback				
Code Quantity Ever	-0.046***	-0.046***	-0.046***	-0.046***
	(0.010)	(0.010)	(0.010)	(0.010)
Code Quantity Log	-0.072	-0.072	-0.072	-0.071
	(0.080)	(0.080)	(0.080)	(0.080)
Non-Code Quantity Ever	-0.009*	-0.009*	-0.009*	-0.009*
	(0.005)	(0.005)	(0.005)	(0.005)
Non-Code Quantity Log	-0.023	-0.023	-0.023	-0.023
	(0.035)	(0.035)	(0.035)	(0.035)
Case Correctness Rate	-0.100***	-0.100***	-0.100***	-0.099***
	(0.010)	(0.010)	(0.010)	(0.010)
Code Acceptance Rate	-0.072***	-0.072***	-0.072***	-0.072***
	(0.008)	(0.008)	(0.008)	(0.008)
Team × Year FE	✓	✓	✓	✓
Decile d–1 activity	✓	✓	✓	✓
Decile d–1 message	✓	✓	✓	✓

Notes: This table reports 2SLS estimates using alternative instrument construction methods, as indicated in the column headers. All specifications include team-by-year fixed effects and developer controls. OLS and reduced-form estimates for the same specification appear in Table 6. Standard errors are clustered at the reviewer level. Column (1) presents the main 2SLS estimates using the residualized leave-out mean leniency score. Column (2) applies limited information maximum likelihood (LIML) with all reviewer dummies as instruments. Column (3) reports modified bias-corrected 2SLS estimates following Kolesár et al. (2015). Column (4)

D.2 Forecast-Based VA Estimator

I estimate reviewer value added (VA) for the full sample using the forecast-based estimator proposed by Chetty et al. (2014). Equation (8) models developer code output Y_{isjt} after receiving feedback from reviewer j, controlling for lagged output $Y_{i,t-1}$, developer characteristics $\mathbf{X}_{it}$, and team characteristics $\mathbf{S}_{st}$. Reviewer effects are captured through a vector of indicators $\mathbf{T_j}$, with coefficients $\boldsymbol{\theta}$ representing reviewer value added. From this regression, I obtain residuals Y^*_{isjt} by subtracting the fitted values excluding reviewer fixed effects, as shown in Equation (9). I then compute $\bar{Y}^*_{jt}$, the average residual for reviewer j in year t, defined in Equation (10). Finally, Equation (11) forecasts reviewer VA γ_{jt} based on the leave-one-year-out residual averages $\bar{\mathbf{Y}}^*_{\mathbf{jt-}}$.

$$Y_{isjt} = \beta_0 + Y_{i,t-1}\beta_1 + \mathbf{X_{it}}'\boldsymbol{\beta_2} + \mathbf{S_{st}}'\boldsymbol{\beta_3} + \mathbf{T_j}'\boldsymbol{\theta} + \varepsilon_{isjt} \tag{8}$$

$$Y^*_{isjt} = Y_{isjt} - [\beta_0 + Y_{i,t-1}\beta_1 + \mathbf{X_{it}}'\boldsymbol{\beta_2} + \mathbf{S_{st}}'\boldsymbol{\beta_3}] \tag{9}$$

$$\bar{Y^*_{jt}} = \frac{1}{n_{jt}} \sum_{i \in \{i:j(i,t)=j\}} Y^*_{isjt} \tag{10}$$

$$\gamma_{jt} = \pi_0 + \bar{\mathbf{Y}^*_{\mathbf{jt-}}}'\boldsymbol{\pi_1} + u_{jt} \tag{11}$$

The forecast-based estimator offers several advantages for estimating reviewer value added. First, instead of including reviewer fixed effects directly in the regression, the model omits them from the fitted values. This yields a modified residual Y^*isjt (Equation (9)) that captures variation not explained by past developer output or covariates. This step reduces attenuation bias caused by measurement error in the covariates. Second, to avoid overfitting and make use of prior information, the model predicts reviewer value added in week t using the average residuals from all other weeks (a leave-week-out approach). This produces $\hat{\gamma}jt$, the best linear predictor of reviewer j's impact in week t. Third, Equation (11) allows the reviewer VA to vary over time.

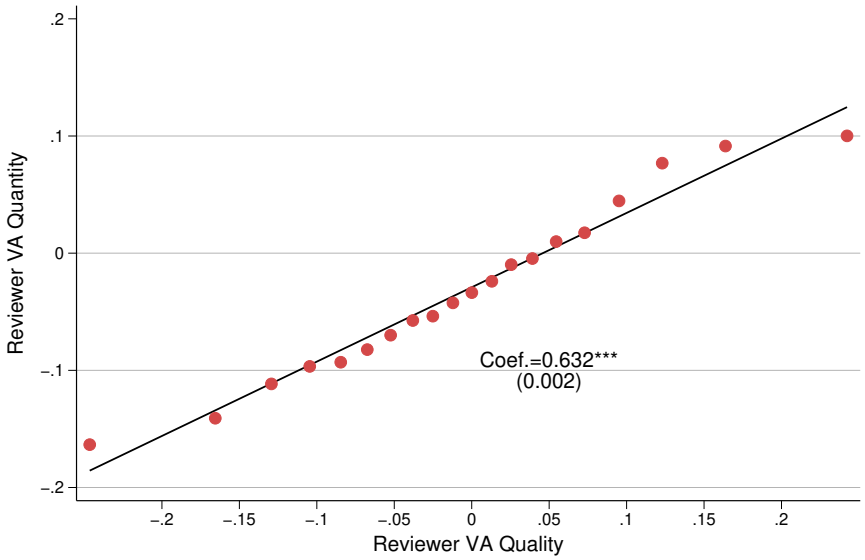
E GitHub - LinkedIn Data Construction

The goal is to build a panel on developers' communications, activities, and employers by merging GitHub with Revelio Labs. I obtain names, employers, biographies, profile images, locations, and emails from the GitHub API. Because profiles are used for career search, developers tend to report accurate details (El-Komboz and Goldbeck, 2024). In the random reviewer assignment sample, about 70% of developers list a LinkedIn profile, personal website, or other social account on GitHub, and many reuse the same photo, which aids identification.

The matching process proceeds in three steps. First, I create direct matches for developers who link their LinkedIn URL on GitHub or vice versa. Second, for unmatched cases, I construct a match pool using names, locations, employment histories, and biographies. To account for naming variations, I match on common nicknames (for example, "Alex" for "Alexander" or "Alexandra"), abbreviated names (for example, "Tina L." for "Tina Liu"), and names with or without middle names (for example, "Tina Jin Liu" simplified to "Tina Liu"). The pool includes exact matches, first name with last initial matches, and matches involving middle names; developers with entirely different names or typos are excluded. Third, I incorporate additional characteristics such as position, firm, and geographic location to identify the best LinkedIn candidate for each GitHub developer. A research assistant reviews all candidates in the pool and selects the best match. I also tested an LLM-based approach (GPT 4o-mini and DeepSeek R1) to select matches, but its performance was inferior to manual review.

F Additional Figures and Tables

Figure F1: Reviewer Value-Added Based on Developer Code Quantity vs. Code Quality

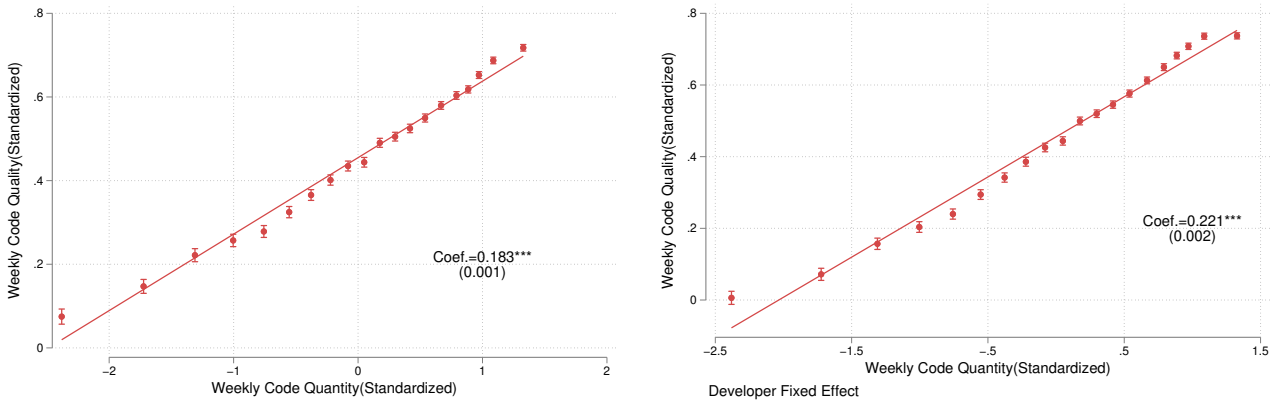


Notes: This figure compares reviewer VA estimates based on two different developer outcomes: code quantity (y-axis) and code quality (x-axis) in the full sample. Reviewer VA is estimated using the forecast-based estimator by Chetty, Friedman, and Rockoff (2014). The fitted line shows a positive and statistically significant relationship between the two measures, indicating that reviewers who increase developer output also tend to improve code quality.

Figure F2: Developer Weekly Code Quantity vs. Code Quality

(a) Without Developer Fixed Effect

(b) With Developer Fixed Effect



Notes: The number of observations is 256,998 in panel (a) and 253,050 in panel (b).

Table F1: Test of Monotonicity

	Split I (Gender)		**Split II (Asian)**		**Split III (Activity)**		**Split IV (Experience)**		**Split V (Followers)**	
	Male	Others	Asian	Others	Less	More	Less	More	Less	More
Panel A. Toxic										
Reviewer Toxicity	0.701***	0.708***	0.727***	0.714***	0.706***	0.697***	0.702***	0.712***	0.701***	0.725***
	(0.015)	(0.046)	(0.046)	(0.018)	(0.014)	(0.028)	(0.016)	(0.023)	(0.015)	(0.034)
F-Stat.	2,087.50	239.38	251.04	1,623.17	2,721.44	632.20	1,866.46	970.15	2,257.34	451.62
Panel B. Positive										
Reviewer Positivity	0.816***	0.789***	0.832***	0.816***	0.813***	0.846***	0.807***	0.845***	0.824***	0.811***
	(0.006)	(0.017)	(0.015)	(0.007)	(0.005)	(0.009)	(0.006)	(0.006)	(0.005)	(0.013)
F-Stat.	18,476.72	2,225.65	2,964.45	13,847.83	26,252.07	8,830.59	16,173.51	17,913.26	28,733.81	4,078.64
Panel C. Constructive										
Reviewer Constructiveness	0.860***	0.866***	0.885***	0.858***	0.867***	0.859***	0.852***	0.885***	0.867***	0.849***
	(0.004)	(0.011)	(0.010)	(0.005)	(0.004)	(0.007)	(0.005)	(0.005)	(0.004)	(0.010)
F-Stat.	38,760.69	5,984.54	7,343.11	28,196.83	47,478.40	16,968.30	35,778.10	28,685.66	55,841.15	7,488.37
N	789,792	89,921	109,912	557,633	829,262	350,017	719,968	459,309	983,909	195,407

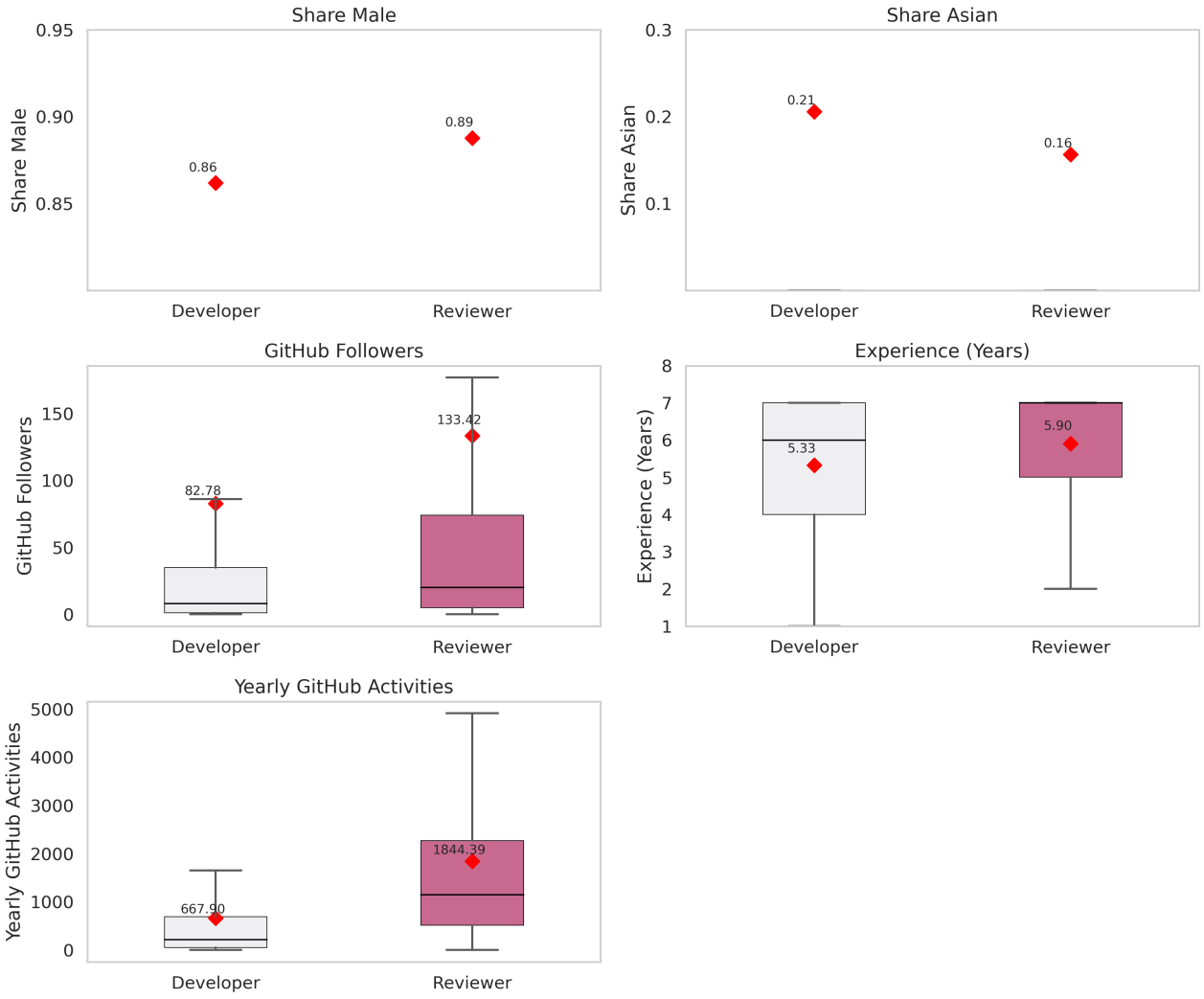
Notes: This table reports the first-stage coefficients and F-statistics from regressions of the reviewer's decision on the instrument across different sample splits. Columns 1–2 separate the sample by gender, Columns 3–4 by race (Asian vs. others), Columns 5–6 by pre-week activity, Columns 7–8 by experience (based on first active year on GitHub), and Columns 9–10 by number of followers. Standard errors in parentheses. Significance levels: * $p < 0.10$, ** $p < 0.05$, *** $p < 0.01$.

Table F2: Assessing Bias in On-Platform Feedback Effects

	Interaction	s.e.(Interaction)	p-value
Toxic Feedback			
Code Quantity Ever	0.199	0.176	0.261
Code Quantity Log	−1.625	1.584	0.305
Non-Code Quantity Ever	0.033	0.101	0.744
Non-Code Quantity Log	0.557	0.637	0.382
Case Correct Rate	0.082	0.202	0.684
Code Correct Rate	0.206	0.165	0.210
Positive Feedback			
Code Quantity Ever	−0.004	0.067	0.949
Code Quantity Log	0.685	0.451	0.129
Non-Code Quantity Ever	0.014	0.037	0.714
Non-Code Quantity Log	0.113	0.202	0.576
Case Correct Rate	−0.000	0.068	1.000
Code Correct Rate	−0.058	0.052	0.263
Constructive Feedback			
Code Quantity Ever	0.033	0.034	0.326
Code Quantity Log	−0.369	0.236	0.118
Non-Code Quantity Ever	−0.014	0.018	0.439
Non-Code Quantity Log	0.251**	0.113	0.026
Case Correctness Rate	−0.012	0.036	0.737
Code Acceptance Rate	0.019	0.026	0.463
Team × Year FE	✓	✓	✓
Decile d–1 activity	✓	✓	✓
Decile d–1 message	✓	✓	✓

Notes: The "Interaction" column reports the coefficient on $I^{\theta}_{im,t} \times C_{im}$. The Co-located dummy C_{im} equals 1 if all team members are in the same country. See Appendix A.3 for details on how team location is constructed. Outcome variables are 1-4 weeks after the developer received feedback. Standard errors are clustered at the reviewer level. * $p < .10$, ** $p < .05$, *** $p < .01$.

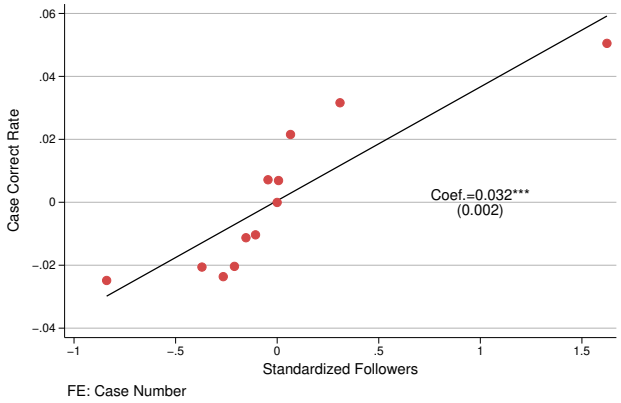
Figure F3: Comparison between Developers and Reviewers



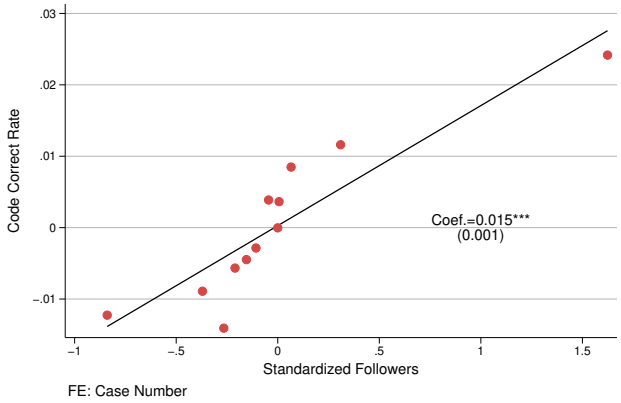
Notes: Each panel compares characteristics between developers and reviewers on GitHub. Developers are defined as those who never serve as reviewers. The sample includes teams that adopt random reviewer assignment. "Share Male" and "Share Asian" report the average share across individuals (red diamonds). "GitHub Followers," "Experience," and "Yearly GitHub Activities" display distributions and mean values (red diamonds) using standard box plots. The black line inside each box represents the median; whiskers extend to 1.5 times the interquartile range. GitHub followers are scraped as of 2023. Experience is calculated as 2024 minus the first year a developer appeared on GitHub. Yearly GitHub activities measure the total number of activities in the most recent calendar year.

Figure F4: Output Quality Validation

(a) Case Correctness Rate



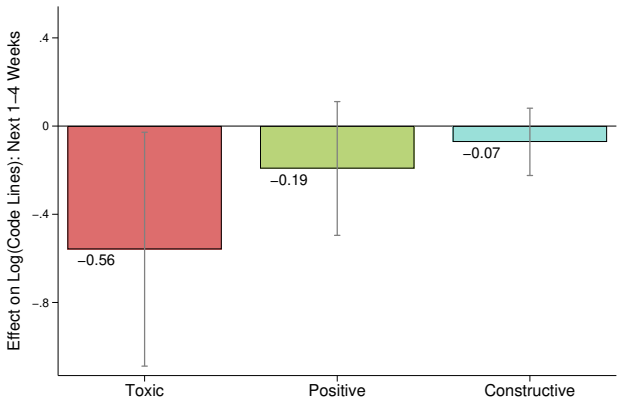
(b) Code Acceptance Rate



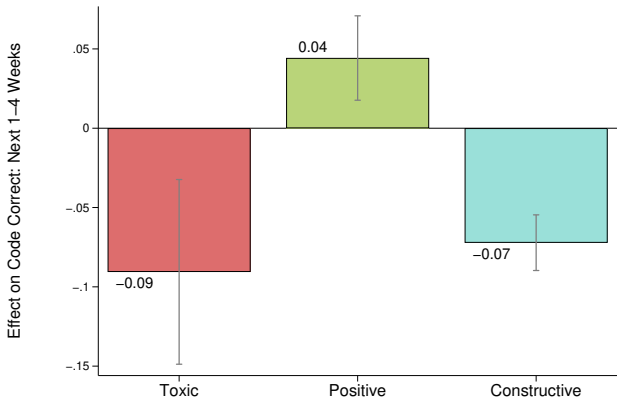
Notes: This figure validates output-quality measures. The x-axis displays each developer's standardized follower count (as a proxy for reputation), and the y-axis shows their average case correctness rate and code acceptance rate as defined in Appendix A.5. The binscatter controls for the total number of cases per developer to account for varying activity levels. We observe that developers with more followers exhibit higher correct rates.

Figure F5: Cluster at Team Level

(a) Code Quantity

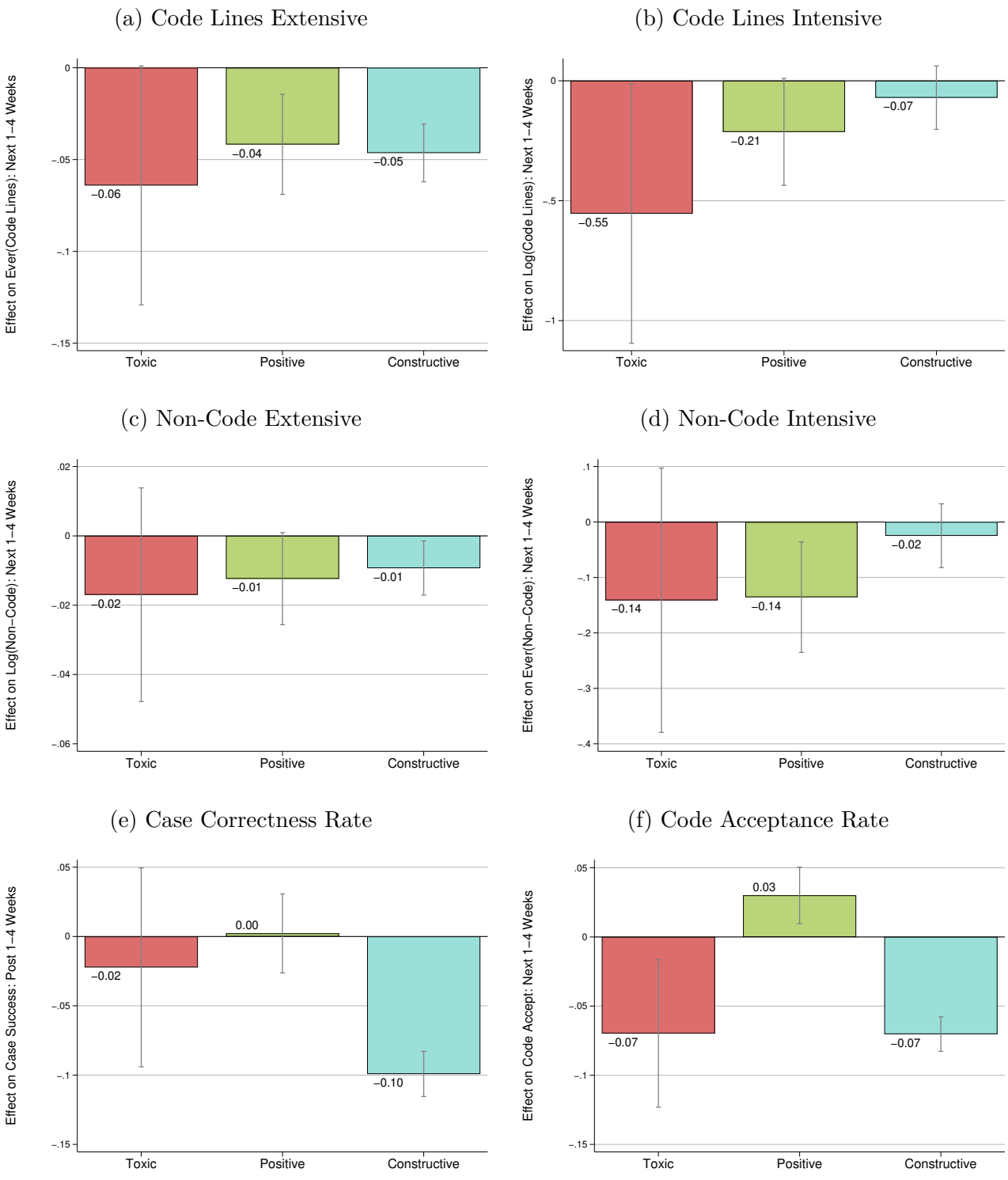


(b) Code Quality



Notes: Panels (a) and (b) show estimates of feedback on code quantity and code quality for developers in the next 1-4 weeks. Standard errors are clustered at the team level. Lines report 90% confidence intervals.

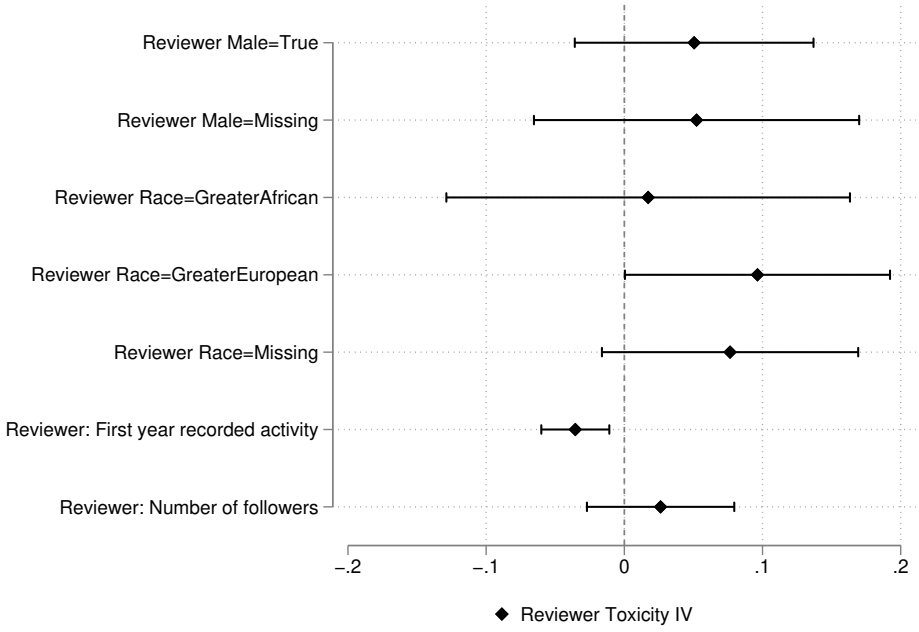
Figure F6: IV Model with Three Feedback Dimensions



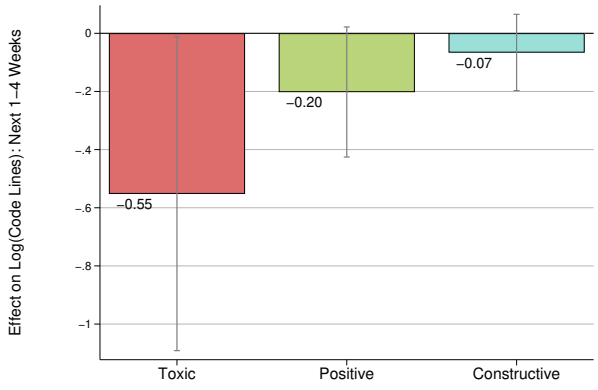
Notes: Baseline sample of cases for random reviewer sample from 2017 to 2023. Controls include all variables listed in Table 6. The results for each type of feedback are from the augmented IV models given by equations (3)–(6). Standard error is clustered at the reviewer level. Lines report 90% confidence intervals.

Figure F7: Controlling for Reviewer Characteristics

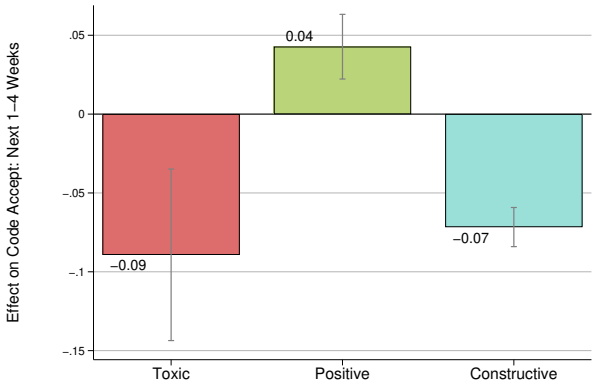
(a) Reviewer Characteristics and Reviewer Toxicity IV



(b) Code Quantity

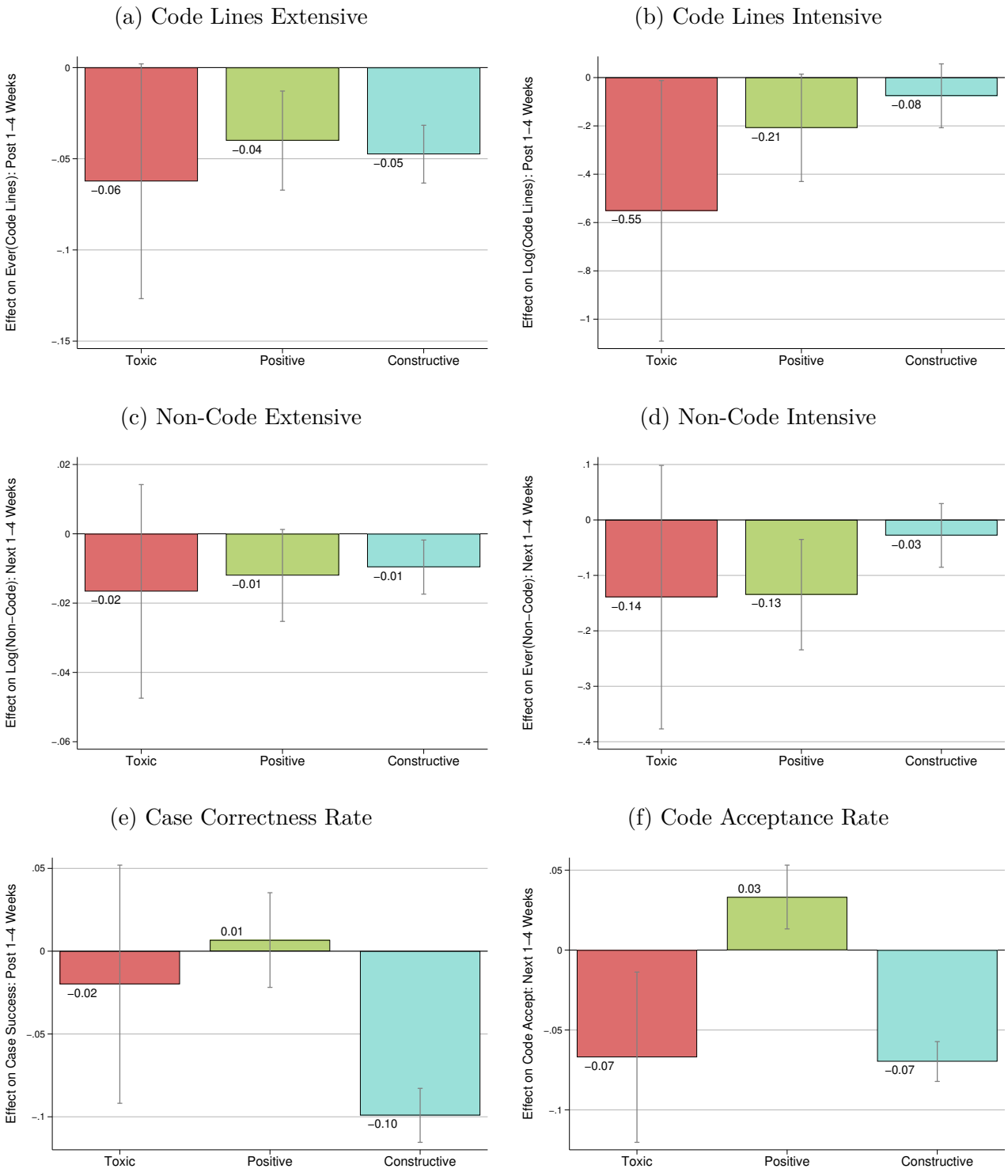


(c) Code Quality



Notes: Panel (a) reports OLS regression coefficients for reviewer toxicity based on standardized reviewer characteristics. The controls include gender, race, the number of followers (in 2023), the first year of GitHub activity (seniority), and reviewer activity in the past year. Standard errors are clustered at the reviewer level. Panels (b) and (c) report the main outcomes of code quantity and code quality when controlling for reviewer characteristics listed in panel (a). The sample and existing controls are the same as in Table 6.

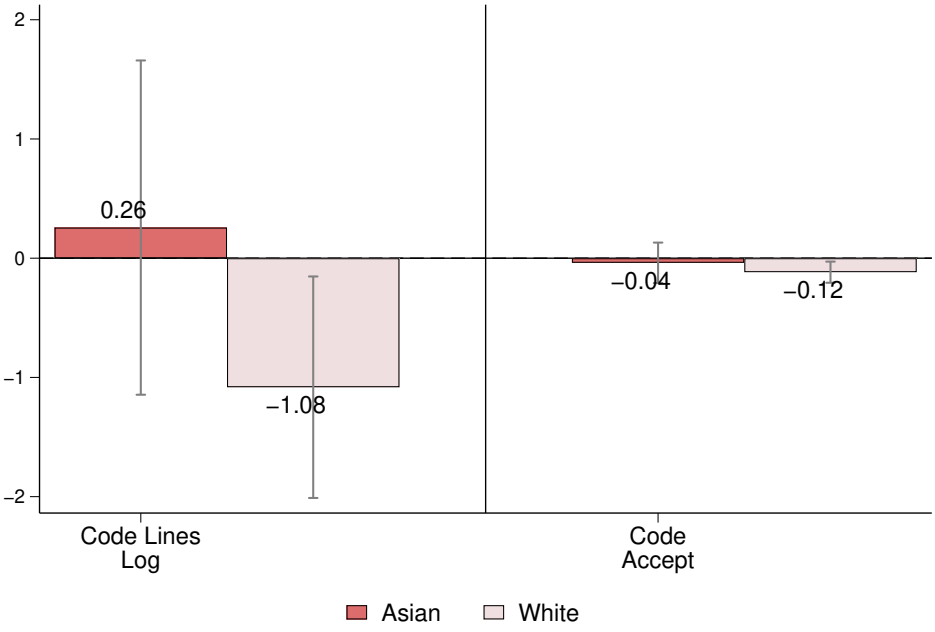
Figure F8: Controlling for Reviewer IV in Feedback Types Other Than the Focal Type



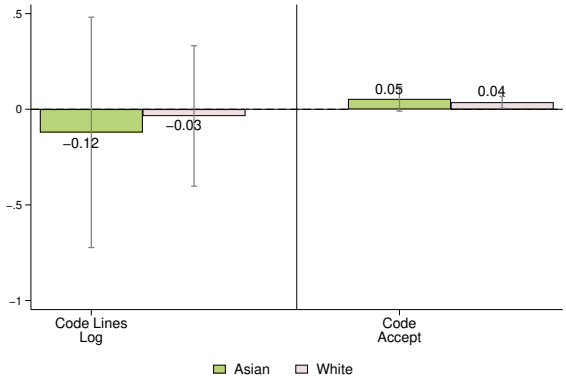
Notes: Baseline sample of cases for random reviewer sample from 2017 to 2023. Controls include all variables listed in Table 6. The results for toxicity controls, including reviewer positivity and constructiveness; positivity controls for reviewer toxicity and constructiveness; and constructiveness controls for reviewer toxicity and positivity. Standard error is clustered at the reviewer level. Lines report 90% confidence intervals.

Figure F9: Heterogeneous Effects of Feedback by Asian vs. White (Next 1–4 Weeks)

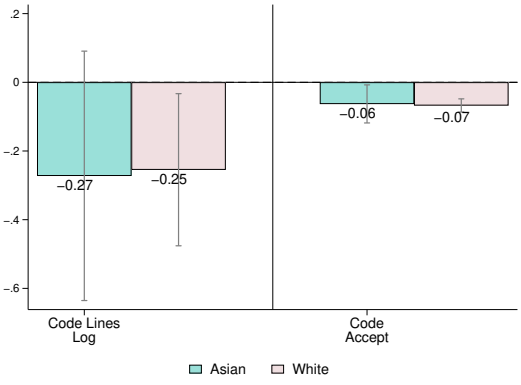
(a) Toxic Feedback



(b) Positive Feedback



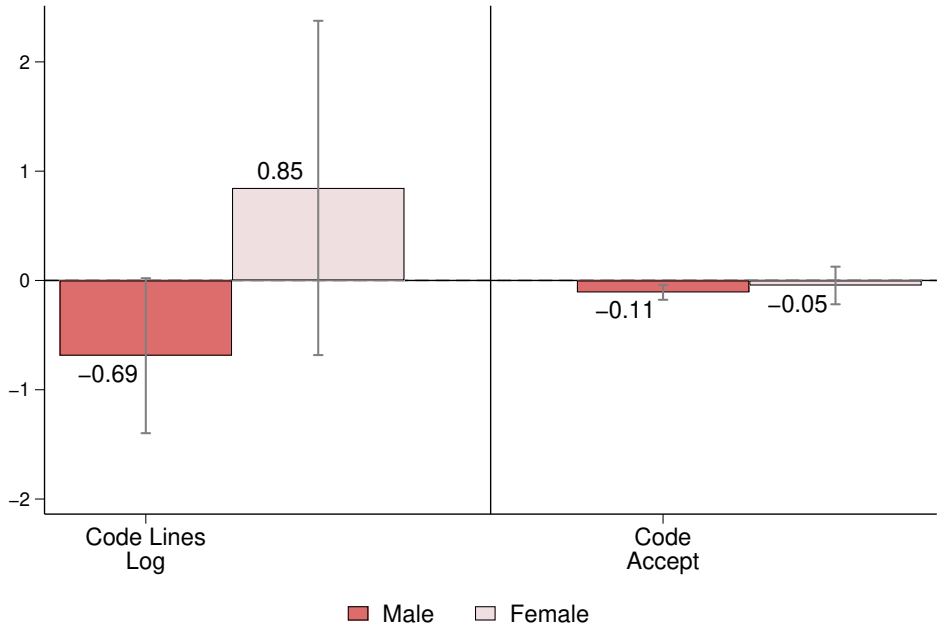
(c) Constructive Feedback



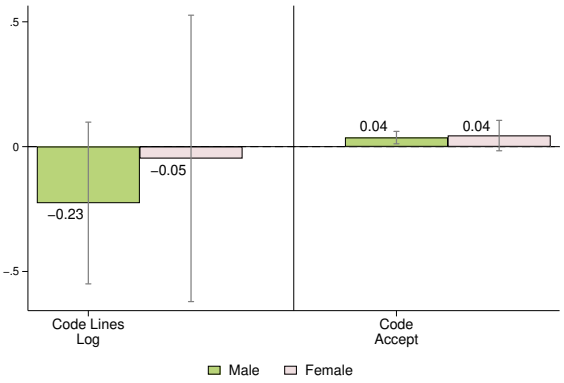
Notes: This figure shows the heterogeneous effects of feedback on developer outcomes in the 1–4 weeks following feedback, by developer race. Dark bars represent Asian developers; light bars represent White developers. Effects are estimated with standard errors clustered at the reviewer level. Vertical lines indicate 90% confidence intervals.

Figure F10: Heterogeneous Effects of Feedback by Male vs. Female (Next 1–4 Weeks)

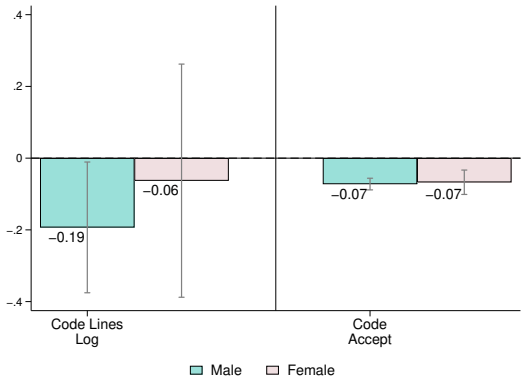
(a) Toxic Feedback



(b) Positive Feedback



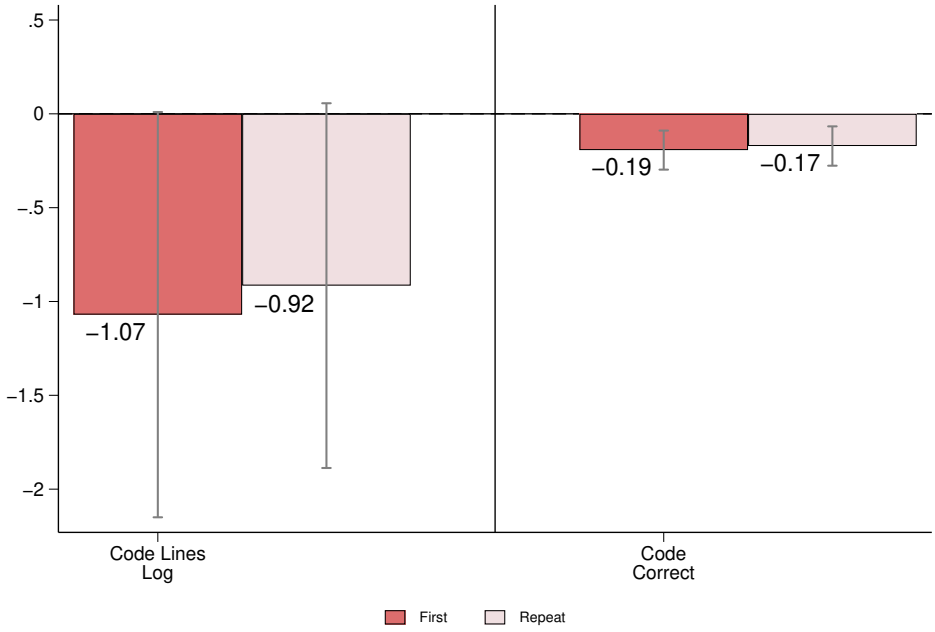
(c) Constructive Feedback



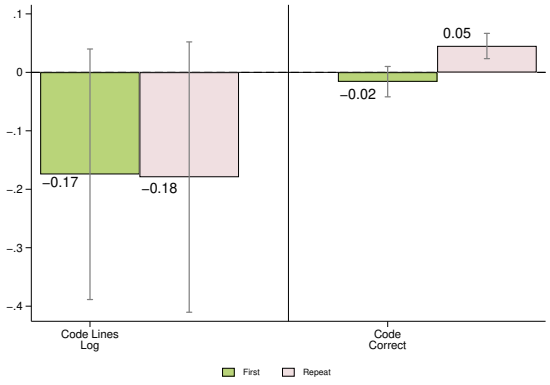
Notes: This figure shows the heterogeneous effects of feedback on developer outcomes in the 1–4 weeks following feedback, by developer gender. Panels (a)–(c) report estimates for toxic, positive, and constructive feedback, respectively. Dark bars represent male developers; light bars represent female developers. Outcomes include code and non-code contributions (ever and log) and whether code or non-code content was retained. Standard errors are clustered at the reviewer level. Vertical lines denote 90% confidence intervals.

Figure F11: Heterogeneous Effects of Feedback by First vs. Repeat Exposure (Next 1–4 Weeks)

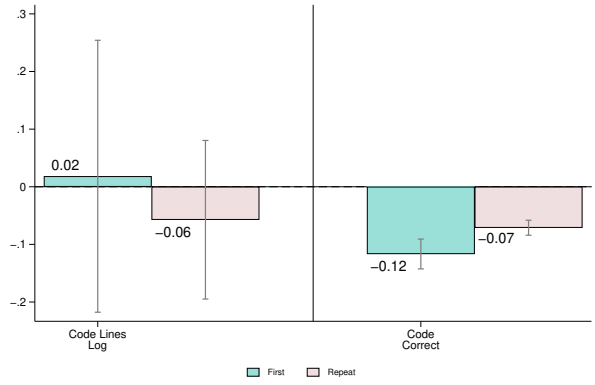
(a) Toxic Feedback



(b) Positive Feedback



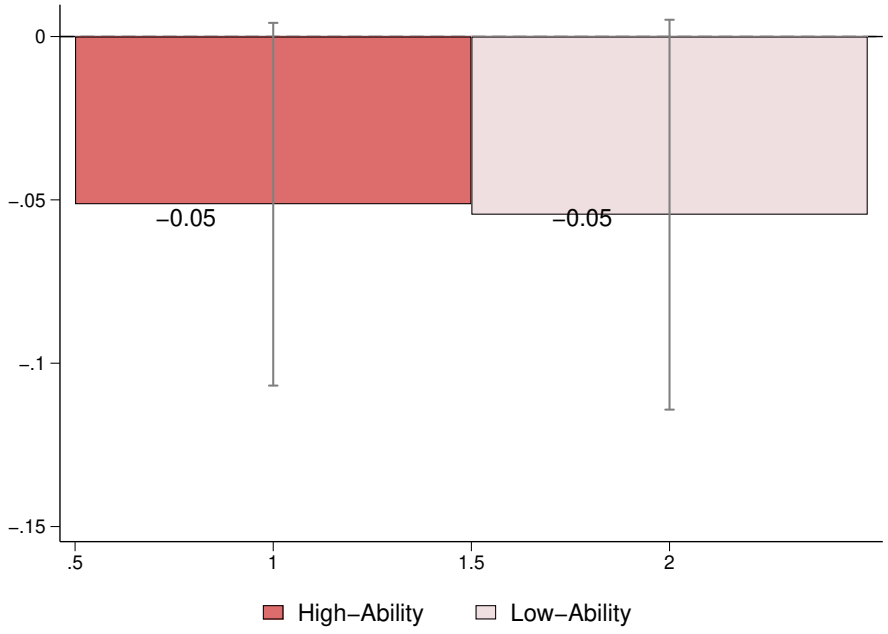
(c) Constructive Feedback



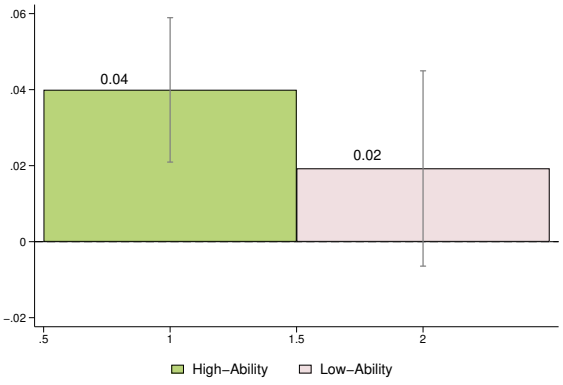
Notes: This figure shows the heterogeneous effects of reviewer feedback by whether the feedback was given on a developer's first case ("First") or later submissions ("Later"). Dark bars represent first-time feedback; light bars represent later feedback. Effects are estimated with standard errors clustered at the reviewer level. Vertical lines show 90% confidence intervals.

Figure F12: Heterogeneous Effects of Feedback by Developer Ability (Next 1–4 Weeks)

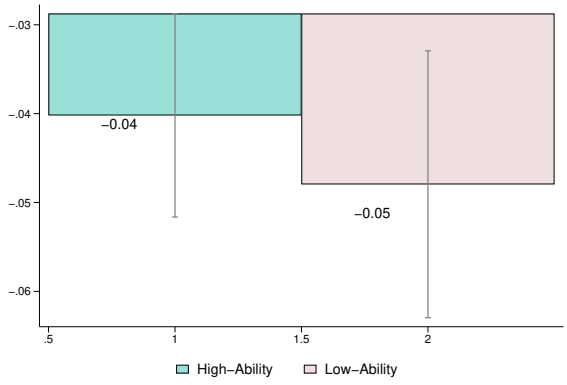
(a) Toxic Feedback



(b) Positive Feedback



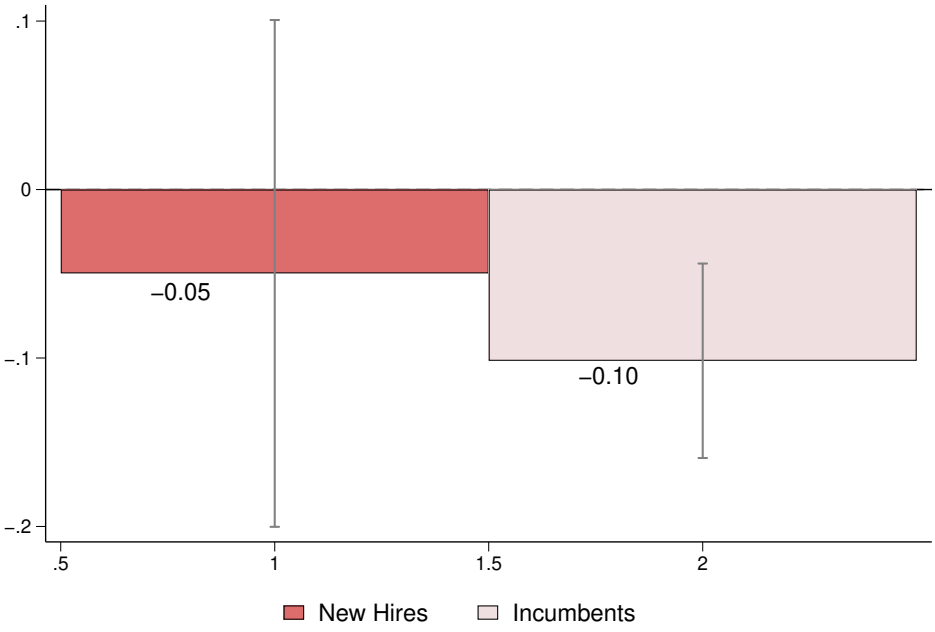
(c) Constructive Feedback



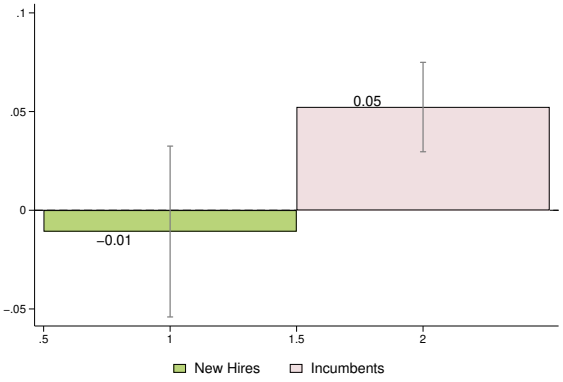
Notes: This figure shows how the effects of reviewer feedback vary by developer ability, defined using lagged code quality. Developers are split into high- and low-ability groups based on their past performance. Dark bars represent high-ability developers; light bars represent low-ability developers. Standard errors are clustered at the reviewer level. Vertical lines indicate 90% confidence intervals.

Figure F13: Heterogeneous Effects of Feedback by New Hires vs. Incumbents (Next 1–4 Weeks)

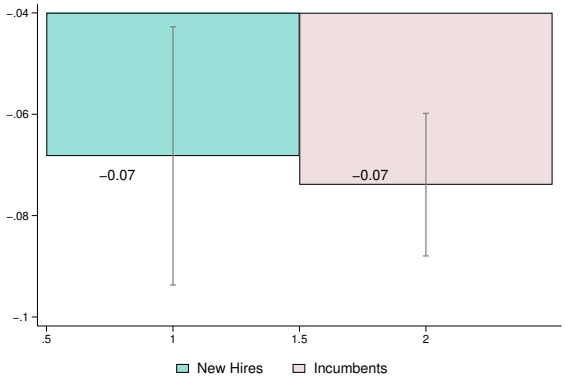
(a) Toxic Feedback



(b) Positive Feedback



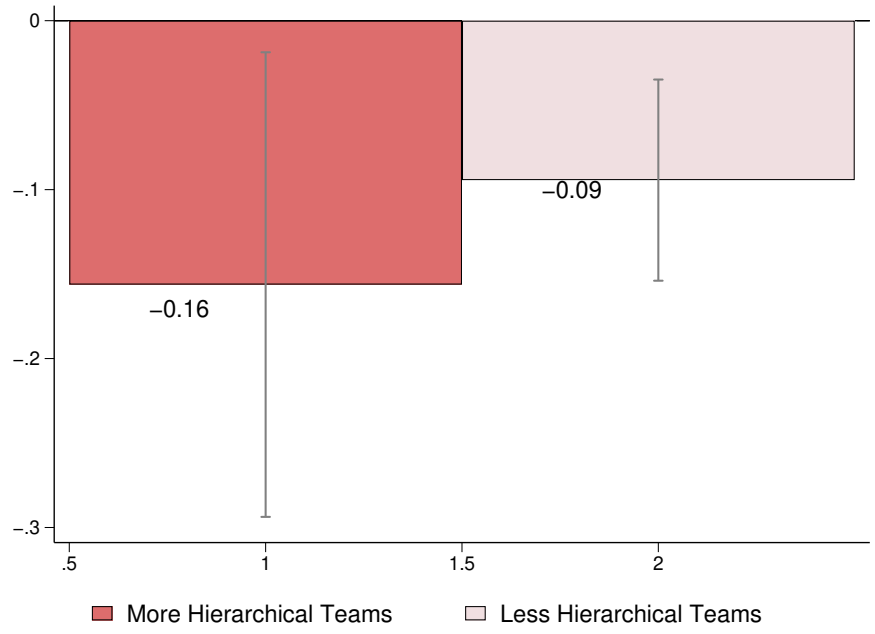
(c) Constructive Feedback



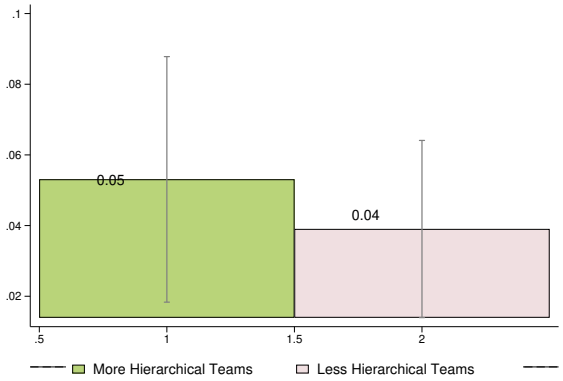
Notes: This figure shows the heterogeneous effects of reviewer feedback by worker status as a new hire (within the first two months on a team) or as an incumbent (longer than two months). Dark bars represent new hires; light bars represent incumbents. Effects are estimated with standard errors clustered at the reviewer level. Vertical lines show 90% confidence intervals.

Figure F14: Heterogeneous Effects in More vs. Less Hierarchical Teams (Next 1–4 Weeks)

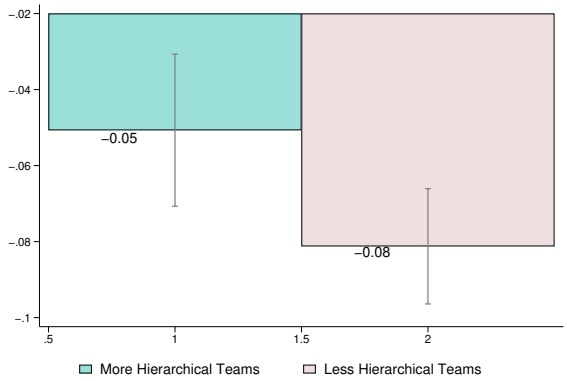
(a) Toxic Feedback



(b) Positive Feedback



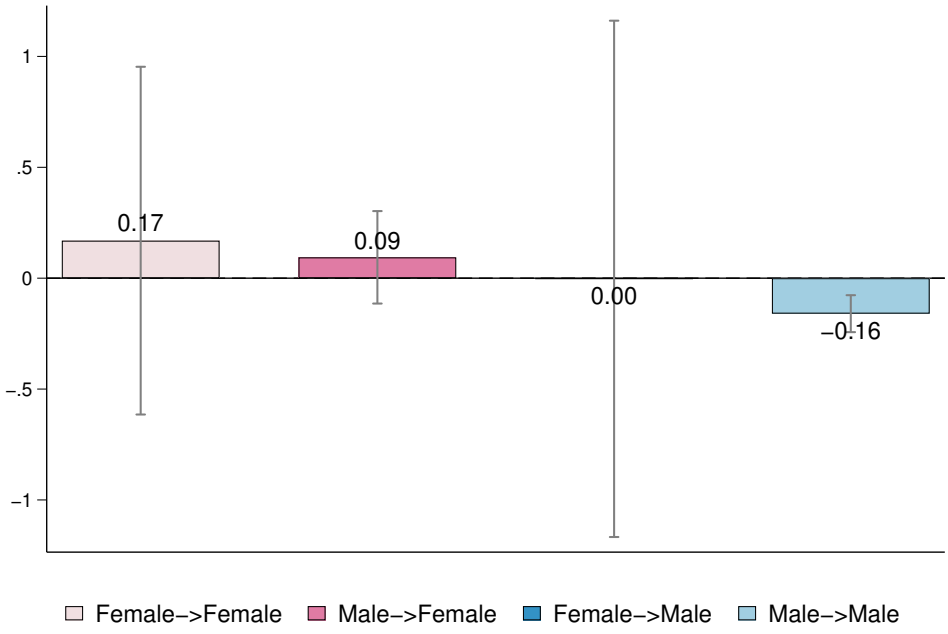
(c) Constructive Feedback



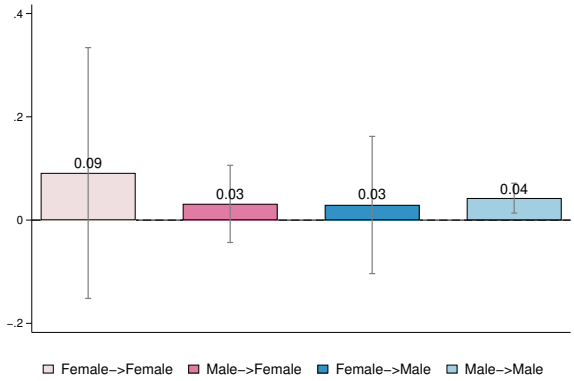
Notes: The figure shows heterogeneous effects by team hierarchical score (see Section 4). Dark bars represent hierarchical teams with a score greater than 0.5; light bars represent flat teams. Effects are estimated with standard errors clustered at the reviewer level. Vertical lines denote 90% confidence intervals.

Figure F15: Gender Match Effects of Feedback on Developer Productivity

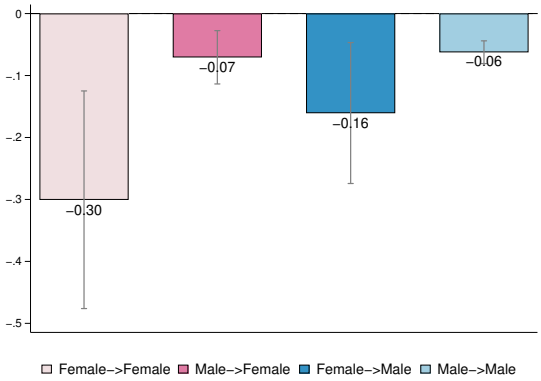
(a) Toxic Feedback



(b) Positive Feedback



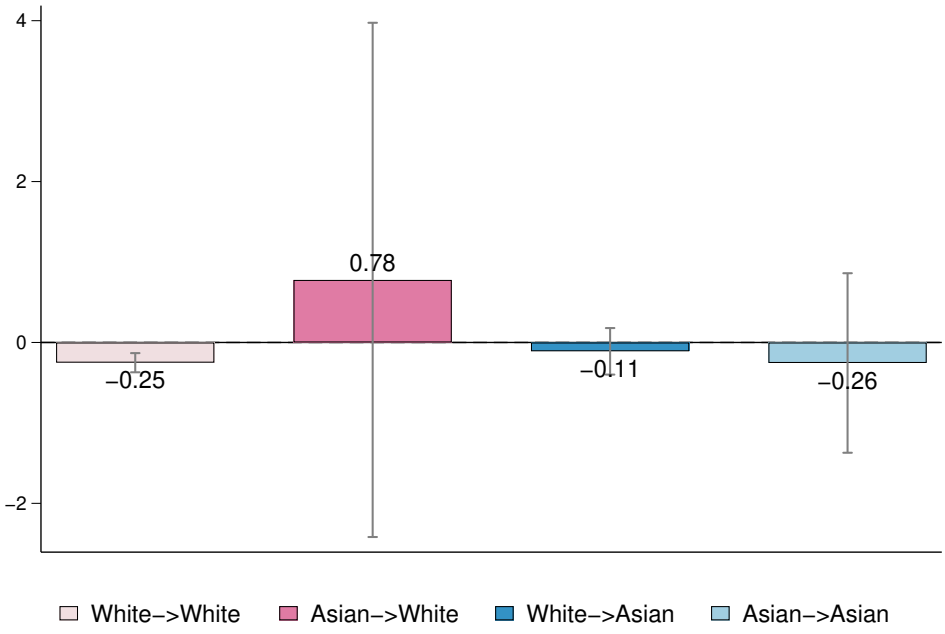
(c) Constructive Feedback



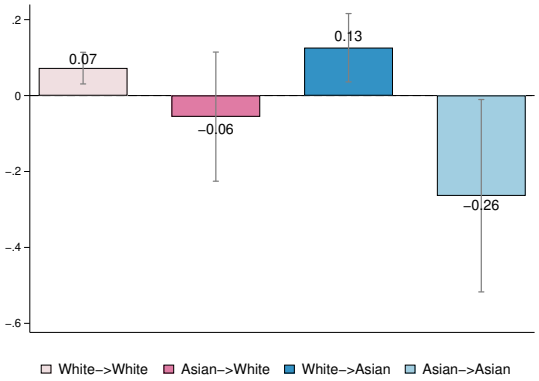
Notes: This figure shows the heterogeneous effects of demographic match between reviewer and developer. For example, "Female → Male" indicates feedback from a female reviewer to a male developer. Standard errors are clustered at the reviewer level. Vertical lines represent 90% confidence intervals.

Figure F16: Race Match Effects of Feedback on Developer Productivity

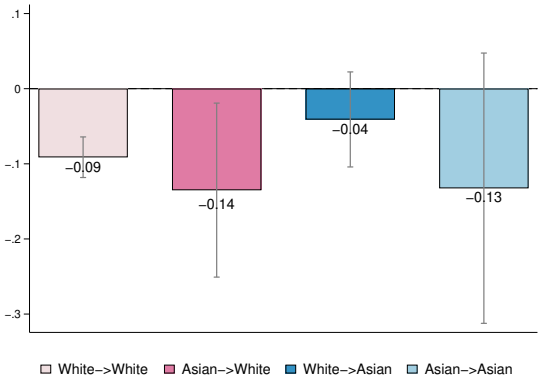
(a) Toxic Feedback



(b) Positive Feedback



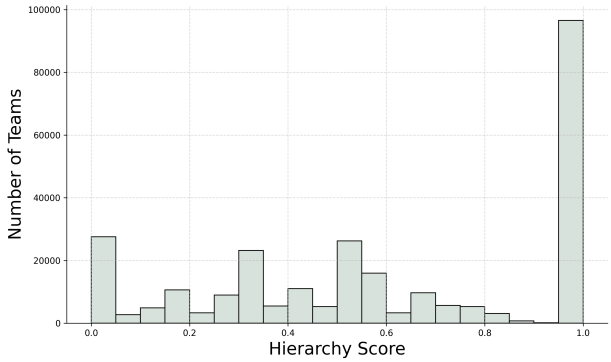
(c) Constructive Feedback



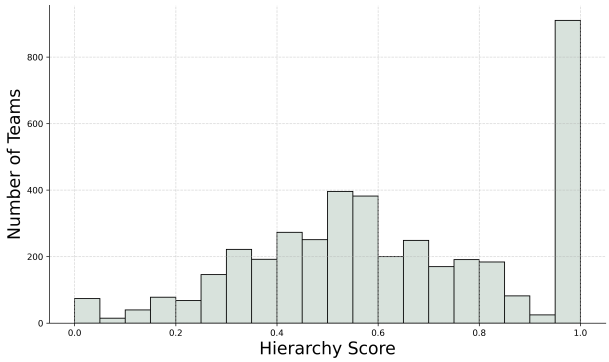
Notes: This figure shows the heterogeneous effects of demographic match between reviewer and developer. For example, "White → Asian" indicates feedback from a female reviewer to a male developer. Standard errors are clustered at the reviewer level. Vertical lines represent 90% confidence intervals.

Figure F17: Distribution of Team Hierarchy Score

(a) Full Sample

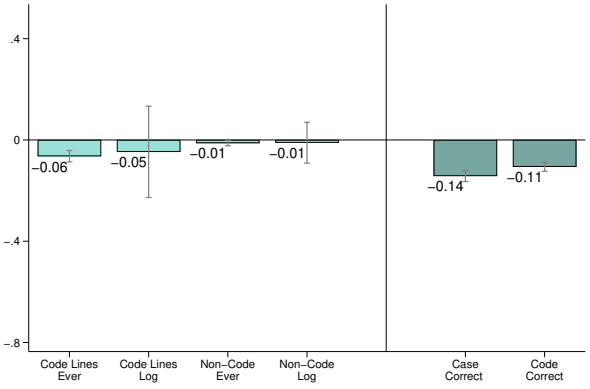


(b) Random Reviewer Sample



Notes: The figure plots the distribution of team hierarchy scores without restricting team size, using data from 2017 to 2023. The full sample covers teams across firms but excludes those with a single case, as they automatically receive a score of 1. The random reviewer sample includes only teams with random reviewer assignment, more than two reviewers per year, and at least 10 cases reviewed per reviewer annually. Higher scores indicate that review responsibilities are concentrated within a small subset of team members, while lower scores reflect a more evenly distributed review structure. The unit of observation is the team–year.

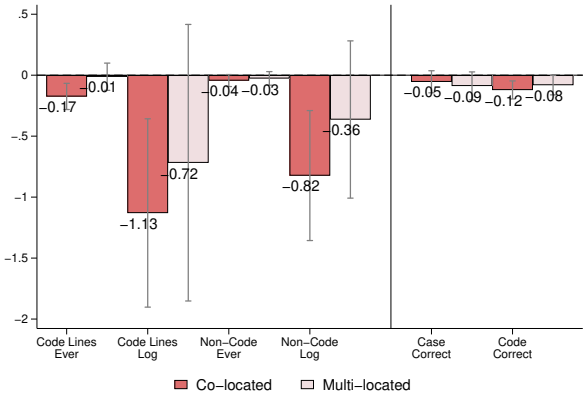
Figure F18: The Effect of Constructive Feedback on Productivity: An Alternative Cutoff



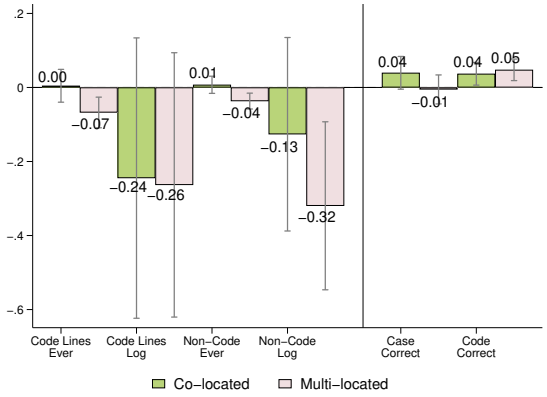
Notes: I construct a binary "constructive" label from the LLM-predicted score, using a cutoff of $\geq$4 in the main analysis in Figure 8d. To assess sensitivity, I lower the cutoff to ≥ 3 —thereby classifying more feedback as constructive—and reestimate the leave-developer-out reviewer constructiveness instrument. Lines report 90% confidence intervals.

Figure F19: Heterogeneous Effects of Feedback by Multi-location Team Structure

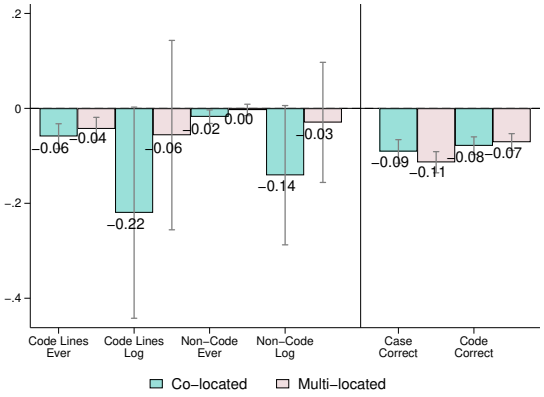
(a) Toxic Feedback



(b) Positive Feedback



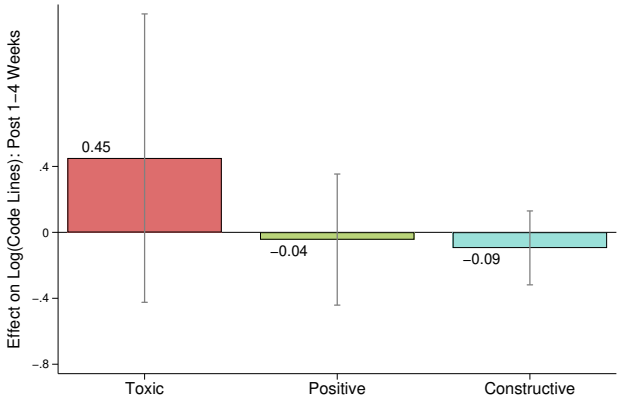
(c) Constructive Feedback



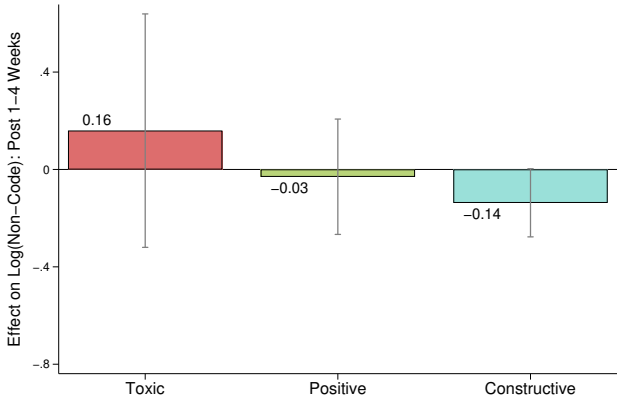
Notes: This figure shows the heterogeneous effects of reviewer feedback by team members in the most represented country, which is below the sample median, indicating more geographically dispersed teams. I estimate the effect in each subsample. Baseline sample of cases for the random reviewer sample from 2017 to 2023. Controls include all variables listed in Table 6. Standard errors are clustered at the reviewer level. Vertical lines show 90% confidence intervals.

Figure F20: Effect of Feedback on Developer Outcomes in Other Teams (Next 1–4 Weeks)

(a) Code Quantity



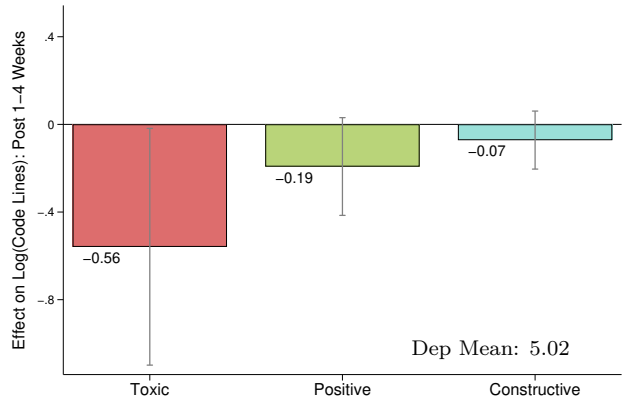
(b) Non-Code Quantity



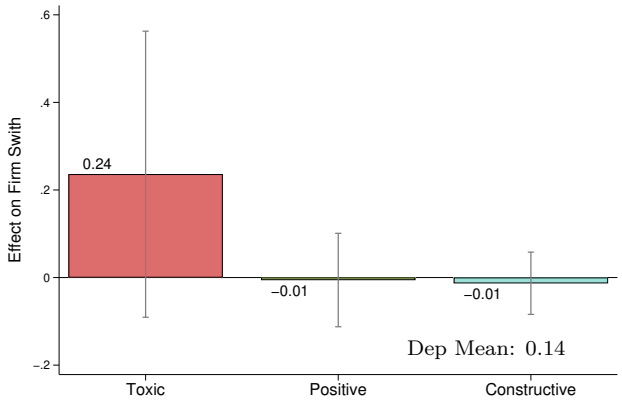
Notes: This figure shows the 2SLS estimates of receiving feedback in a focal team on a developer's productivity in *other* teams during the following 1–4 weeks. Standard error is clustered at the reviewer level. Lines report 90% confidence intervals.

Figure F21: Effect of Feedback on Developer Outcomes Within a Team

(a) Code Quantity (Next 1–4 Weeks)



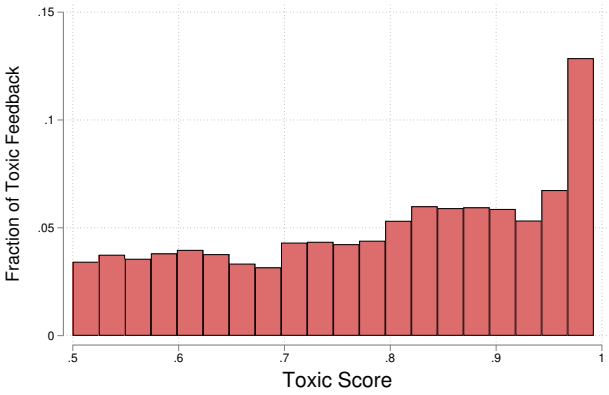
(b) Firm Switch (Next 1-2 Years)



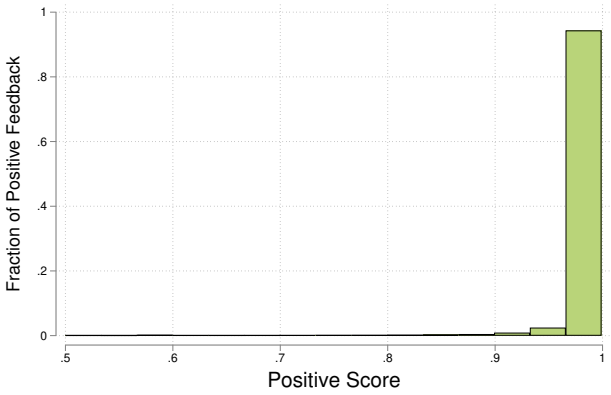
Notes: This figure reports 2SLS estimates of the effect of a given feedback type on developer outcomes within the same team during weeks 1–4 after the feedback. The sample includes teams with random reviewer assignment, at least three reviewers per year, and reviewers with at least 10 cases per year. The industry-exit and firm-switch outcomes come from LinkedIn; for these two outcomes, estimates use a 10% random subsample of the random-reviewer sample used in panels (a)–(d). Standard errors are clustered by reviewer, and lines show 90% confidence intervals.

Figure F22: Distribution of Predicted Feedback Probability

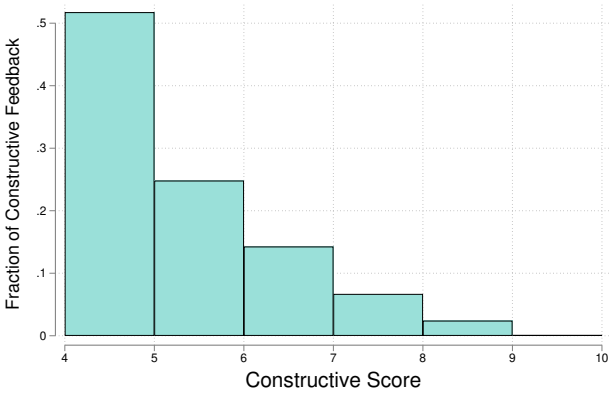
(a) Conditional on Toxic Feedback



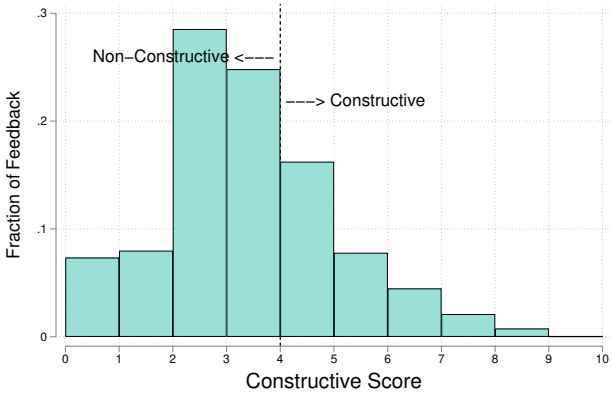
(b) Conditional on Positive Feedback



(c) Conditional on Constructive Feedback



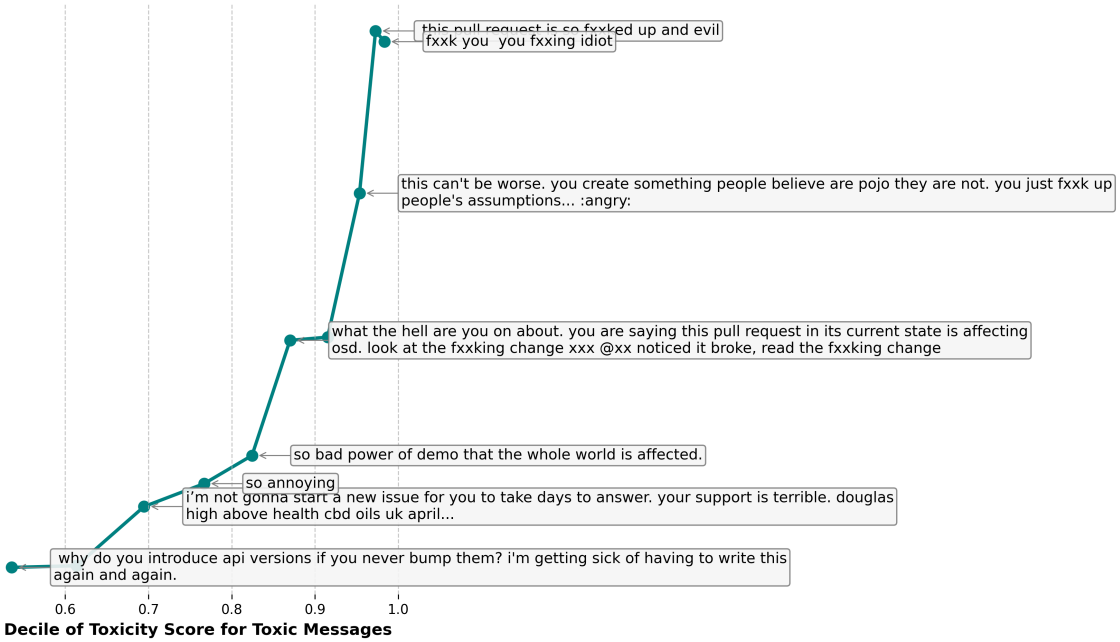
(d) Constructive Feedback Cutoff



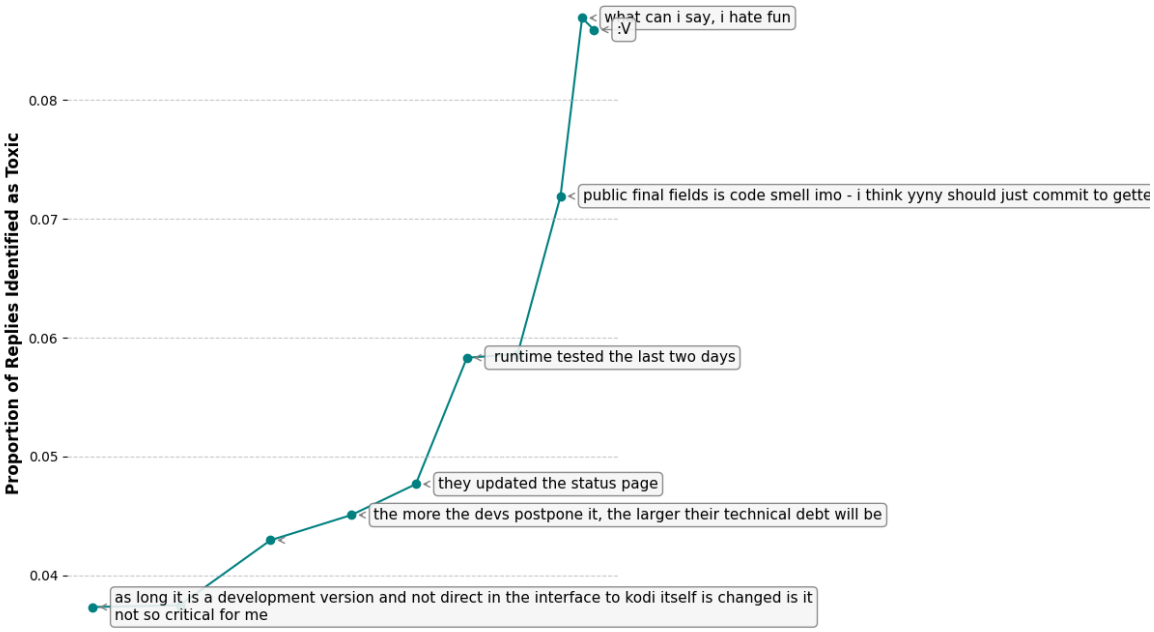
Notes: These figures illustrate the distribution of predicted probability scores for each feedback dimension, based on all feedback messages written by code reviewers in the random reviewer sample. Figures F22a and F22b report score distributions for toxic and positive feedback, respectively, based on binary labels assigned by a pre-trained classifier. Figure F22c shows the distribution of predicted constructive scores generated by GPT-4o-mini. Figure F22d illustrates the classification of feedback as constructive or non-constructive using a cutoff on the GPT-based score: messages above the threshold are labeled constructive, while those below are labeled non-constructive. As a robustness check, I replicate the analysis with an alternative cutoff of 3 (rather than 4); the results remain qualitatively unchanged (see Figure F18).

Figure F23: Toxic Feedback and Replies

(a) Toxic Feedback



(b) Corresponding Replies



Notes: Figure F23 displays randomly selected examples of toxic feedback and their replies in the full sample. These interactions often occur in contexts where senior members provide feedback or evaluations to juniors, and developers who receive toxic comments frequently respond by explaining their actions or decisions.

G Prediction

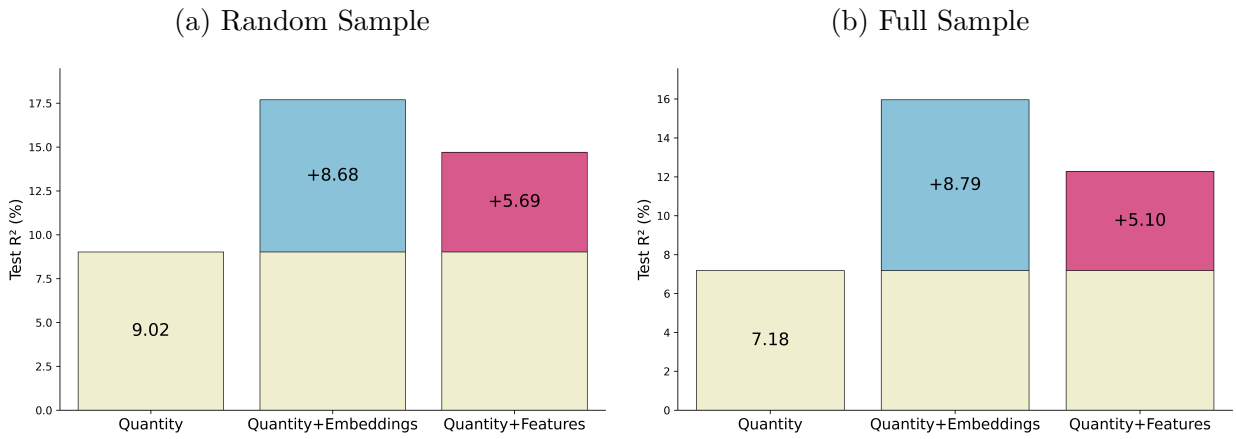
To predict reviewer value-added, I apply the Extreme Gradient Boosting (XGBoost) algorithm. For the continuous outcome, the model uses the squared error loss (*reg:squarederror*) and is evaluated using root mean squared error (RMSE). I randomly split the data into a training set (80%) and a hold-out test set (20%). Model training and tuning are performed on the training set using a five-fold cross-validation approach.

Hyperparameters are selected using a two-stage procedure: a random search explores a wide parameter space, followed by a grid search that refines the best-performing regions. Repeat the tuning process separately for each predictor set. Figure G1 examines how feedback explains variation in reviewer value-added, measured by changes in developer code quality before and after assignment. Panel (a) uses the random-assignment sample, while Panel (b) uses the full sample, including both random and non-random assignments. Reviewer value-added is estimated using the forecast-based estimator of Chetty et al. (2014).

I compare three sets of covariates: (i) feedback quantity, measured as the total and average number of messages per case; (ii) feedback quantity combined with 768-dimensional text embeddings that capture the semantic content of messages; and (iii) feedback quantity combined with selected features, defined as the weekly shares of toxic and positive feedback per reviewer.

The results show that feedback content explains substantial variation in reviewer value-added. In the random sample, feedback quantity alone accounts for about 9 % of the variance. Adding embeddings increases explanatory power by roughly 18% percentage points. Using only the selected features captures most of this improvement. The full sample yields similar patterns. These findings highlight that the tone and content of feedback are salient predictors of reviewer quality.

Figure G1: Feedback Characteristics Explain Variation in Reviewer Value Added



Notes: This figure shows how different aspects of reviewer feedback explain the variation in reviewer value added (VA), measured by its impact on developer code quality. Each bar reports the test R^2 from models using different inputs: *Quantity* is the number of feedback messages written; *Quantity+Embeddings* adds 768 semantic dimensions of the text; *Quantity+Features* includes the weekly shares of toxic, positive, and constructive feedback per reviewer. Values on the bars indicate the improvement in explanatory power relative to the quantity-only model.