Artificial Intelligence in the Firm: Bottlenecks in Software Production

Fiona Chen
Harvard University
Job Market Paper

James Stratton
Harvard University

Current: November 14, 2026
First: January 7, 2026

Current version here

Abstract

We examine the impacts of generative AI on productivity, output, and employment in software engineering. Firm production requires combining complementary inputs, across which AI has differential productivity effects. Inputs that are difficult to automate can become "bottlenecks," constraining output and shaping labor demand. We study these dynamics using a novel proprietary dataset from an engineering analytics platform, covering 445M work events — GitHub coding activity, Jira issues, and Google Calendar events — for 720,542 workers at 655 firms. We estimate the effects of two generations of AI coding tools — assistants and agents — by exploiting variation in timing of firm adoption. AI substantially increases code output but has small effects on final software output and no detectable effect on employment. Code review becomes the bottleneck. Code waits longer to reach a reviewer, spends longer in active review, and receives more requested changes. We build and calibrate a model of firm production and AI adoption, then use it to simulate AI's long-run impacts under continued firm adjustment and technological progress, predicting growth in output and declines in total employment, with smaller effects if the review bottleneck persists.

Contact: fionachen@g.harvard.edu and jstratton@g.harvard.edu. We are immensely grateful to Erik Brynjolfsson, Raj Chetty, Larry Katz, Mandy Pallais, and Jesse Shapiro for their guidance and support throughout this project. We are also indebted to the Jellyfish team, including Nicholas Arcolano, Lena Chretien, David Gourley, and Andrew Lau, for their partnership and their expertise on AI and software engineering. Finally, we would like to thank many individuals for their feedback, including Lukas Althoff, Josh Angrist, David Autor, Nick Bloom, Deirdre Bloome, Sydnee Caldwell, Bharat Chandar, Mihai Codreanu, Jessica Dai, Mert Demirer, Rebecca Diamond, Catia Farquharson, Claudia Goldin, Jamie Gracie, Andy Haupt, David Holtz, Ruru Hoong, Anders Humlum, Alex Imas, Chad Jones, Xianruo Kang, Arjun Ramani, Aakaash Rao, Alana Renda, Anna Russo, Clare Suter, Nico Thor, Neil Thompson, and Oscar Volpe; as well as seminar participants at the Berkeley Artificial Intelligence Research (BAIR) Lab, Google, Harvard Economics, Harvard Kennedy School, INFORMS MSOM, the Machine Learning in Economics Summer Conference at the University of Chicago, MIT Computer Science and AI Laboratory (CSAIL), MIT FutureTech, OpenAI Conference, Opportunity Insights, Stanford Digital Economy Lab, Stanford Labor/Public Workshop, and the Society of Labor Economists (SOLE) Conference. This research was supported by funding from the Stripe Economics of AI Fellowship; Harvard Kennedy School's James M. and Cathleen D. Stone Program in Wealth Distribution, Inequality, and Social Policy; and the Schmidt Sciences AI at Work Program. The conclusions of this study are those of the authors and do not represent the views of Jellyfish. The results are based on data that has been aggregated and anonymized. The manuscript was reviewed by the data partner solely to verify that it does not disclose proprietary information, as required by the research agreement; it was not reviewed for its scientific content.

1 Introduction

Artificial intelligence (AI) has advanced rapidly in recent years, prompting significant excitement as well as concern about its labor market consequences (Kwa et al., 2025; Epoch AI, 2026; Sajadieh et al., 2026). Yet there remains substantial uncertainty over the extent and trajectory of this transformation. This new technology has the potential to deliver significant economic growth and generate widespread job displacement. However, its impacts will depend not only on its technical capabilities, but also on how it is integrated into existing production and market structures (Davidson, 2021; Briggs & Kodnani, 2023; Chui et al., 2023; Autor, 2024; Acemoglu, 2024; Korinek et al., 2026).

We study the impacts of AI on productivity, output, and employment in software engineering. In particular, we focus on how the structure of firm production shapes these effects. Initial indicators of the technology's impacts have demonstrated its promise, with numerous studies documenting that AI raises productivity on specific tasks across domains (e.g., Noy & Zhang, 2023; Peng et al., 2023; Brynjolfsson, Li, & Raymond, 2025). However, these productivity gains may not translate into commensurate changes in firms' final output or employment. Production processes generally combine complementary inputs, some of which are more difficult to automate than others. These inputs can thus become "bottlenecks", limiting the pass-through of productivity gains to final output and leading firms to reallocate labor (Kremer, 1993; Aghion et al., 2017; Brynjolfsson et al., 2018; Acemoglu & Restrepo, 2019; Gans & Goldfarb, 2026; C. Jones & Tonetti, 2026). However, empirically identifying where bottlenecks emerge and how firms respond to them is difficult, as doing so requires highly granular data on internal firm production processes.

We leverage a novel proprietary dataset from Jellyfish (JF), an analytics platform that companies use to understand the activity of their software engineering teams. Our data allows us to measure details of the software production process, covering 445 million work events of 720,542 workers across 655 firms, from January 2021 to August 2026. JF combines data from industry-standard software engineering tools, and we use four of these tools: (1) code version control systems, such as GitHub, which track engineers' code; (2) issue management systems, such as Jira, which measure engineers' progress on and completion of software tasks and projects; (3) scheduling tools, such as Google Calendar, which capture meeting activity and time allocation; and (4) generative AI usage at the firm and worker level, for GitHub Copilot, Cursor, and Claude Code.

Software has been at the frontier of advances both in AI's technical capabilities and in organizational adaptation: AI labs have invested significant resources in training models to complete coding tasks, such that tools have become highly competent, and adoption has been widespread (Kinder et al., 2024; Eloundou et al., 2024). For instance, the frontier AI labs state that AI is already writing 80% of their code, and they are reorganizing workflows in response (The Anthropic Institute, 2026).

Given the rapid advances in these tools' capabilities, we examine the impacts of two generations of AI coding tools — AI coding assistants and AI coding agents. First released in late 2022, AI coding assistants provide workers with localized code edit suggestions. Advances in AI model capabilities led to the release of agentic coding tools in late 2024, which have the capacity to semi-autonomously complete tasks of greater length and complexity.

To structure our empirical analysis, we begin by building a model of firm production and AI adoption, in which production requires complementary inputs and AI has differential productivity effects across inputs. Specifically, software development consists of two stages, to which the firm allocates labor inputs: code writing and code review. AI adoption affects production through two channels. First, it increases the productivity of coders, increasing the volume of code requiring review. Second, it decreases the quality of code entering review, increasing the review effort required per unit of code and decreasing review productivity. Both channels can generate a "bottleneck" in the review stage, limiting the pass-through of the coding productivity shock to final output and shaping firms' labor demand towards the bottleneck task. Although we describe our model in the context of software development, its ideas are generalizable to a broad range of settings in which production requires complementary inputs that are differentially affected by AI. In particular, the structure of "implementation" versus "management" or "review" is nearly ubiquitous across knowledge work settings — and our model demonstrates that even if AI adoption increases the speed of implementation, output will be limited by the speed of management or review, and AI can even create spillovers that further slow down these surrounding processes.

We next test the predictions of our model empirically. We estimate the effects of AI adoption using a staggered difference-in-differences design, exploiting variation in the timing of firms' adoption of AI coding assistants and agents. Our identification strategy relies on the assumption that absent adoption, earlier- and later-adopting firms would have followed similar trajectories in outcomes. In practice, the technology procurement process typically involves a number of steps — including information gathering, security and legal approval, and small-scale piloting (Rapoport et al., 2025; IBM, 2024). The lengths of these processes differ significantly across firms, in a way that is plausibly orthogonal to firms' growth trajectories, and we leverage this variation for our identification strategy.

With this design, we begin by studying the effects of generative AI tools on measures of intermediate code output, using three outcomes: lines of code added; commits, which represent incremental saved changes to code; and pull requests, which represent larger code submissions for review. AI assistants lead to moderate increases in code output — 12 months after adoption, firms exhibit an increase of 30.4% in lines of code, 16.8% in commits, and 20.9% in pull requests. AI agents lead to larger additional increases in all three measures — 44.0% for lines of code, 36.0% for commits, and 35.7% for pull requests opened. These estimates are consistent with recent work on the effects of

generative AI tools on coding productivity (Cui et al., 2024; Hoffmann et al., 2024; Song et al., 2024; Sarkar, 2025; Demirer, Musolff, & Yang, 2026).

Does the extra code written translate into increases in firm-level software output? In principle, extra code may not generate new useful output for the firm: code may be subject to errors; and even if it is not subject to errors, surrounding production steps may constrain its deployment. We use two measures of output: resolution of Jira issues, which occur once a body of coding work is deployed to production; and resolution of Jira epics, which represent completion of larger projects or features. The effects from AI assistants are near zero, at −1.5% on issues and 3.4% on epics. The effects from AI agents are slightly larger — 13.7% on issues and 18.0% on epics — with the effect of agents on issues resolved being borderline significant. One concern is that AI adoption may lead to an increase in the size or complexity of issues, such that the raw issue count would understate true output growth. We address this by analyzing the text descriptions of Jira issues, using machine learning tools to predict the length of time it would take for an engineer to resolve them. We use two methods — first, using an LLM to label issues with their assessed length; and second, using a supervised learning method to predict the outcome label of time to resolution. We find no evidence of an effect on the average predicted length of each issue. In turn, we observe effects on final software output that are much smaller in magnitude than those of the intermediate code output.

If firms do not adjust on the margin of increased output, they may adjust by cutting employment. Many companies have stated that AI tools are already affecting demand for software engineers, with some firms announcing layoffs due to AI (e.g., Kessler, 2023; Roose, 2025), and others announcing increases in hiring (e.g., Stanley, 2025). We assess the effect on engineering employment, measured by the number of active workers at each firm on Jellyfish. We cannot attribute significant employment changes to AI, though our estimates are noisy. With AI assistants, we estimate an effect of 3.0%, ruling out changes below −4.7%. For AI agents, we estimate an effect of −2.6%, ruling out changes below −15.6%.

Taken together, our results show that while AI tools substantially raise measures of intermediate code output, these gains do not translate into proportional changes in software output or employment. How can we reconcile these findings?

We turn to examining the structure of software production. Writing code is only one step in a multi-stage production process: upstream, managers must plan and scope features; downstream, written code must be reviewed and tested before it can be deployed to production. We impute time usage from the Jellyfish data and validate this imputation with a survey of 100 Prolific engineers and external survey estimates of time allocation, finding that engineers spend 30–40% of their time writing code; the remainder goes to these complementary tasks. When AI raises productivity in code writing, complementary steps may serve as bottlenecks. To provide color to our results, we perform

qualitative interviews with 21 workers at software companies. Anecdotally, engineers have pointed to these surrounding tasks as important bottlenecks that limit the gains from AI:

> *People could make a huge number of commits quickly, but code review was still the bottleneck. On one project, only one senior engineer was allowed to approve changes because he was the expert on the codebase. He was so bandwidth constrained that he could only review my code once a week. After some outages that were caused by AI-generated commits, there were also new rules requiring senior engineer approval for AI-generated code, which created even more review bottlenecks.*

Our data provides us with particularly granular insights into the code review process, which we examine as one source of a bottleneck. Though firms do not typically hire separate code writers and code reviewers, workers are also not perfectly substitutable across these stages. In particular, as evidenced by the description above, workers typically only review code that they are familiar with (82.8% of reviews, compared to 17.2% if reviewers were assigned at random); additionally, code written by a senior engineer is more often reviewed by another senior engineer (75.7% of reviews for senior coders, compared to 32.8% for junior coders).

We find evidence of a bottleneck in this stage: after adoption of AI agents, the average time from the code's review submission to its approval increases by 56.1%. This change represents both an increase in the time that code waits in a "backlog," by 59.9%, and an increase in the time after code has begun active review, by 58.5%. The latter is consistent with a decline in review productivity, i.e., an increase in the review intensity. The share of code changes that receive at least one review comment increases by 13.1%, and the number of comments on each code change increases by 38.6%. For AI assistants, we do not observe significant effects on any of these margins, which we attribute to the fact that the effect on the amount of code written is comparatively small in magnitude. This bottleneck is not eliminated by adoption of AI tools in the review process — 79.1% of firms have adopted AI code review tools, but still largely keep humans in the loop, with the overwhelming majority of reviews and comments being written by humans.

Finally, we bring our empirical estimates to our model of software production. First, we estimate the model using minimum distance to recover effects that we cannot observe directly. We observe the impacts of AI adoption on intermediate code output and on employment, but not on coding or review productivity — i.e., the rate at which labor inputs produce code and reviews — or on how workers change their time allocation between coding and review. AI agents raise coding productivity by 73.7% and slightly lower review productivity (−7.0%), and firms shift labor from coding to review: coding labor changes by −22.7% and review labor by 43.4%. Notably, the effect on coding productivity is larger than the effect on code output, because the increase in final output combined with the decline in review productivity implies that some labor is reallocated away from coding towards review.

Second, with the estimated model, we simulate the potential future impacts of AI under two types of scenarios: continued firm adjustment and continued technological advancement. The simu-

lations help to highlight the importance of three factors in shaping the longer-run impacts of AI on economic outcomes: the trajectory of technological advancement, the software product market, and the persistence of the review bottleneck. First, the size of the software product market will govern how much firms can expand output, and in turn how much they will cut employment. We begin by characterizing a threshold for the product demand elasticity, above which an adopting firm sees a net increase in employment and a larger increase in output, and below which employment falls. We obtain a threshold of $\varepsilon = 3.68$. Existing estimates of demand elasticities for software products range from 1.50 to 3.73 (Ghose & Han, 2014; Derdenger, 2014; Ghose & Sundararajan, 2005), nearly all below this threshold. At the mean of these estimates, $\varepsilon = 2.27$, our model predicts an increase in output of 16.1% and a decline in employment of 13.2%. Next, we combine our estimates with forecasts of the trajectory of technological advancement to simulate possible growth and employment trajectories over the next five years. In particular, the simulations highlight the importance of whether review remains a bottleneck — i.e., whether AI raises review productivity as quickly as coding productivity. If review productivity stays fixed, the model predicts that output rises by 57.8% by 2031 and employment falls by 10.3%, with output leveling off once the review constraint binds. If review productivity keeps pace, output rises by 341.4% and employment falls by 76.2%.

Related literature. Our project contributes to several literatures. First, a long literature examines the impacts of new technologies on labor market outcomes. Most recently, this literature has turned to the effects of AI. These papers document widespread adoption of AI (Chatterji et al., 2025, 2026; Yotzov et al., 2026), as well as productivity gains across settings (Choi & Schwarcz, 2023; Dell'Acqua et al., 2023; Chen & Chan, 2024; Kim et al., 2024; Noy & Zhang, 2023; Otis et al., 2024; Roldan-Mones, 2024; Brynjolfsson, Li, & Raymond, 2025; Dell'Acqua et al., 2025; Dillon et al., 2025; Manzoor et al., 2025). Several papers also examine the effects on firm output and economic growth. At the firm level, AI adoption has been linked to growth in sales, productivity, and market value (Babina et al., 2024; Czarnitzki et al., 2023; Eisfeldt et al., 2023); at the aggregate level, a growing literature models its potential effects on economic growth, with predictions ranging from modest productivity gains to explosive growth (Aghion et al., 2017; Nordhaus, 2021; Trammell & Korinek, 2023; Erdil & Besiroglu, 2023; Acemoglu, 2024; Besiroglu et al., 2024; Filippucci et al., 2024; C. I. Jones, 2024; Korinek & Suh, 2024; C. Jones, 2026; Davidson et al., 2026; Korinek et al., 2026). Finally, several papers examine AI's employment effects, with some finding employment changes, particularly losses for entry-level workers (Brynjolfsson, Chandar, & Chen, 2025; de Souza, 2025; Klein Teeselink, 2025; Lichtinger & Hosseini Maasoum, 2025), and others finding no employment effect (Humlum & Vestergaard, 2025; Gimbel et al., 2025; Iscenko & Millet, 2026). We contribute to this literature in two ways. First, we estimate the effects of firm-level adoption of AI tools, which allows us to capture the aggregate impacts

of AI, accounting for incomplete take-up and heterogeneous utility across workers. Furthermore, it allows us to AI's productivity effect, and connect it to changes in firm output, employment, and firm organization. Second, we provide estimates of these effects across generations of AI coding tools, then use those to simulate future impacts, allowing us to examine how the impacts of AI evolve as the technology improves.

Second, a broader literature examines how productivity gains — including from new technologies — translate to changes in firm output and employment. Although the empirical literature on AI's effects on output and employment is growing, it is often less clear why productivity gains do — or do not — pass through to these downstream outcomes. A large theoretical literature provides structure to understand this pass-through. While new technologies can create productivity gains in individual tasks, complementary tasks can serve as "bottlenecks," constraining changes in output and shaping the reallocation of labor (e.g., Kremer, 1993; B. Jones, 2025; Demirer, Horton, et al., 2026; Gans & Goldfarb, 2026; C. Jones & Tonetti, 2026). Relatedly, task-based models characterize how new technologies reshape the allocation of labor across tasks, determining whether they augment labor, substitute for labor, or create new work (e.g., Acemoglu & Restrepo, 2018; Acemoglu, 2024). Finally, a large related literature in operations and product development characterizes how production systems respond when one stage speeds up (Thomke, 1998, 2003), including how queues and waiting times grow (Little, 1961; Kingman, 1961; Little, 2011), how lower-quality output and rework may slow production (So & Tang, 1995; Smith & Eppinger, 1997a,b; Loch & Terwiesch, 1998), and how firms can reallocate capacity in response (Jordan & Graves, 1995; Zychlinski & Gurvich, 2026). Limitations in detailed micro-data make it difficult to directly test the empirical importance of these theoretical mechanisms. Our paper contributes to this literature by providing direct empirical evidence on the importance of these channels.

Third, a related literature examines how organizations adapt to technology adoption. This work argues that realizing the benefits of new technologies requires complementary organizational investments — including changes to workflows, incentive structures, and other managerial practices — rather than technological adoption alone (e.g., Bresnahan et al., 2002; Aral et al., 2012; Brynjolfsson et al., 2018; Seamans & Raj, 2019; Raj & Seamans, 2019; Dixon et al., 2021; Anthony, 2021; Berg et al., 2023; Anthony et al., 2023; Agrawal et al., 2024; Beane & Anthony, 2024). Relatedly, a management and marketing literature examines how firms organize the process of transforming ideas into successful products, emphasizing the integrated roles of marketing capabilities, product architecture, and organizational structure (e.g., Henderson & Clark, 1990; Griffin & Hauser, 1996; Baldwin & Clark, 2000; Hauser et al., 2006; MacCormack et al., 2006; Mu, 2015; Kyriakopoulos et al., 2015). More recently, this literature has begun examining how generative AI reshapes marketing content and competition in product markets (Guha et al., 2021; Cillo & Rubera, 2025; Exner et al., 2025; Goldberg & Lam,

2025). We contribute to these literature by providing insights into the effects of AI on the product development process in software engineering, including how firms should respond organizationally.

Finally, within these literatures, several papers study the impacts of AI on software engineering. Studies generally find widespread adoption of AI coding tools across occupations and tasks (Johnston et al., 2026). Many have also found that these tools improve worker productivity, measured by coding output and task completion time (Peng et al., 2023; Cui et al., 2024; Song et al., 2024) — though some report negative effects (Becker et al., 2025). A subset of studies have examined the impacts on work patterns, demonstrating shifts in individual work activity and AI usage patterns (Hoffmann et al., 2024; Yeverechyahu et al., 2024; Sarkar, 2025; Sarkar & Melas-Kyriazi, 2026). The most closely related paper is Demirer, Musolff, & Yang (2026). Using data on open-source coding repositories, Demirer, Musolff, & Yang (2026) show that GitHub Copilot adoption substantially increases lines of code written and commits, but has much smaller effects on pull requests and product releases. Demirer, Musolff, & Yang (2026) attribute this incomplete pass-through to human bottlenecks in the production process. Relative to this paper, our main contribution is to examine the effects of AI adoption in a firm setting — with many of the firms in our sample producing enterprise software — in which features of the production process and organizational structure may shape effects. Our setting allows us to more directly identify the nature of bottlenecks — including explicitly using the granularity of our data to document a bottleneck in the review stage, and to characterize the way that firms reorganize in response.

The rest of the paper proceeds as follows. In Section 2, we describe our data and institutional details about software engineering and AI, including details about the data classification and AI take-up. In Section 3, we describe the model. In Section 4, we describe the effects of AI on productivity, output, and employment. In Section 5, we provide evidence on production bottlenecks. In Section 6, we estimate our model and simulate the future impacts of AI. In Section 7, we conclude.

2 Data and institutional context

This section summarizes the data used in our analysis and provides background on software engineering and AI coding tools.

2.1 Data sources

2.1.1 Jellyfish

Our primary source of data is Jellyfish ("JF"), an analytics platform used by firms to understand the activity of their software engineering teams. Firms use JF to monitor software activity and project

progress, and to understand engineers' productivity, including in relation to their use of AI coding tools. To provide these analytics, JF integrates data on engineers' activity from several tools. We make use of four main sources throughout our analysis.

We analyze a sample of data from JF clients who have consented to their data being used for research, between January 2021 and August 2026. There are 720,542 workers across 655 firms (Table 1). For each firm, Jellyfish is able to access data from a tool through the tool's API integration, which the firm must agree to using.

Software production typically follows well-defined and documented processes (Pressman, 2010; Atlassian, 2026). First, teams determine which software changes to implement — this process typically involves integrating customer requests with internal product and engineering objectives. Second, engineers implement these tasks by writing code. Third, teammates will review and test the proposed code. This process typically involves tests that verify the correctness of the code and its integration with the broader code base, as well as its functionality from the user's perspective. Finally, once approved, the code is deployed to production. Our data provides insights into each stage of this process. Figure 1 visualizes this workflow and the data generated at each stage.

Issue tracking data. Software teams employ "issue tracking" systems to organize, assign, and monitor engineering work.[1] Work is typically organized hierarchically. Related work items are grouped within "projects", which often contain one or more "epics" representing large features or initiatives. Epics are in turn decomposed into individual "issues," which represent discrete units of engineering work that can be assigned to engineers.

Each issue can include a text description, links to related issues, and the issue category (e.g., "Task", "Bug", or "Service Request"). The text description of each issue is typically rich: in our analysis sample, the median length of the description is 580 characters. We provide two examples of the Jira interface in Appendix Figure A1. Panel A displays an example of a team's Jira board, which provides an overview of its issues, including their statuses. Panel B displays an example of an individual Jira issue, including its associated fields.

Engineers track progress on tasks and projects through Jira. Once the code has been written, reviewed, tested, and deployed, a team member marks the issue as "resolved" — i.e., complete. In turn, we view completion of a Jira issue as marking the end of a discrete unit of work. Once all of the issues associated with an epic have been completed and deployed, a team member marks the epic as "resolved". This interpretation is consistent with prior academic literature in computer science (e.g., Ortu et al., 2015; Lenarduzzi et al., 2019; Lüders et al., 2022), and the industry-standard use of Jira; indeed, Jira's product description states that tasks "typically represent individual work items such as

[1]Jellyfish integrates data on several issue management systems, including Jira and Microsoft Azure DevOps. For simplicity, we adopt the language of the most common issue management system globally, Jira.

big features, user requirements, and software bugs" (Jira, 2025). Our qualitative interviews similarly validate this interpretation of the data. One junior software engineer describes the workflow in Jira as follows:

> *Work typically moves through the Jira statuses of 'Implementation', 'Review', 'QA Testing', and then 'Release'. For a task to get marked as completed, it not only has to be merged, but also deployed to production.*

Issue management systems are industry-standard: Atlassian, Jira's owner, claims that four-fifths of Fortune 500 firms use Jira (Farquhar, 2019). Our analysis sample includes 73,139,708 issues, for an average of 5.96 per worker-month.

Code version control data. Software engineering teams employ "version control" systems to record, organize, and coordinate changes to a shared codebase. Industry-standard version control systems share a common structure.[2]

Engineers write code on their local machines. As they update their code, they "commit" changes to their branch to incrementally save progress. When work is ready to be incorporated into the collective codebase, the engineer creates a "pull request". This pull request is then reviewed, typically by a manager or teammate. If needed, the reviewer may submit a request for changes and leave comments detailing the needed changes. Then, the engineer and reviewer will iterate on these changes. If the review process is successfully completed, the code is then incorporated into the shared repository ("merged"); otherwise, the code is not incorporated into the shared repository.[3] We provide two examples of the interface for GitHub in Appendix Figure A2. Panel A displays the interface for committing code. Panel B displays the interface for reviewing and commenting on a pull request.

Version control systems are ubiquitous in software engineering teams: for instance, the world's largest version control platform, GitHub, is used by around 90% of Fortune 100 firms (GitHub, 2025). JF integrates data from several version control platforms, including GitHub, GitLab, and Bitbucket.

Our analysis sample includes 297,586,102 commits, for an average of 24.27 per worker-month. We observe the metadata associated with each commit, including the time of the commit, the identity of the engineer making the commit, and the text associated with the commit. We do not observe the code content of the commit. We also observe 54,142,996 pull requests, for an average of 4.41 per worker-month. Each observation of a pull request includes metadata on the details of the code changed — files changed, lines of code added and deleted — as well as information on the review process — changes requested, comments, merge status.

[2]Some of this terminology depends on the precise version control system used. We adopt the terminology of GitHub, the world's largest platform for version control.

[3]As a note, code may be merged into the codebase but not yet deployed. As a result, we will view the pull request and Jira issue resolution as separate outcomes in the production process.

We use commits and pull requests as related but distinct measures of coding activity. Pushing a commit corresponds to making an update to code; creating a pull request represents the submission of a discrete unit of work for review. In practice, the correlation between these two units of work is around 0.929 at the firm-month level, indicating that workers tend to make more pull requests in time periods in which they are pushing more commits, but the frequency of the two updates is very different: in our analysis sample, workers make 5.50 commits for every pull request.

Google Calendar data. For a subset of 192 firms, JF receives data from engineers' Google Calendar accounts. Google Calendar is a scheduling tool: users create calendar events and share them with one another. Each calendar event includes information on the individuals who are invited, individuals who accept versus decline, text description, and meeting length. In our analysis, we use the Google Calendar data to assess relative time allocation between coding and non-coding work. Our analysis sample covers 20,315,197 events, for an average of 6.59 per worker-month.

HR data. Firms may also input HR information into the JF platform. This information includes the job titles of workers and the organizational chart of the firm. We observe HR information for a minority of workers in our sample.

2.1.2 Revelio

We obtain data on workers' and firms' characteristics by merging the JF data to data from LinkedIn collected by Revelio Labs, a data provider which scrapes worker profiles and updates the database weekly. We first hand-match the analysis sample of JF firms with the list of firms whose data are collected from LinkedIn.

Revelio provides information on firms' industries and LinkedIn descriptions, as well as the overall employment of the firm — i.e., the number of employees including both engineers and non-engineers — measured as the number of LinkedIn profiles connected to the firm.

2.1.3 Qualitative interviews

Alongside the quantitative analysis, we conduct a series of qualitative interviews to support the paper's findings. We interview 21 software engineers, engineering managers, product managers, and related stakeholders across a range of firms, seniority levels, and company types. We use these interviews for two purposes: to assess the plausibility of the identification assumptions described above, and to interpret the mechanisms behind our estimates. All interviewees consented to the inclusion of quotes from their interviews in this paper. Appendix Section C reports the interview material in full.

2.2 Description of analysis sample

Our analysis sample consists of the Jellyfish client firms that satisfy three restrictions. First, the firm must have data present for a sufficiently long period of our analysis timeframe that we are able to estimate the effects of either AI assistants or agents. Second, the firm must have opted in to its data being used for research. Third, the firm must have provided data from both Jira and GitHub, which are our two main data sources.

We provide a description of the firms in our analysis sample in Table 2. The average firm employed 1,676.6 total workers (measured using the Revelio merge, SD 5,537.8) and 248.9 active engineers (measured as active workers in Jellyfish, SD 384.8) in January 2023. The distribution is also right-skewed, with a minority of very large firms. Firms are also generally well established: the average firm is 19.36 years old as of January 2023 (SD 21.60). Notably, our sample largely consists of firms that are medium-to-large and reasonably well-established, excluding smaller and newer start-ups. However, these larger firms constitute the majority of software engineering activity in the U.S.: the Census Bureau's Statistics of U.S. Businesses documents that firms with over 100 employees constitute 88% of software industry employment and 92% of software industry revenue (U.S. Census Bureau, 2025).

The firms span a broad range of business models and industries. We use an LLM to classify each firm's revenue engine from its LinkedIn description into one of three categories: firms that sell a software license, such as a tool used by finance companies or hospitals ("SaaS license", 49.3% of firms); firms whose revenue is a cut of the activity they intermediate or carry rather than a license fee, such as digital marketplaces or platforms ("digital service operator", 29.5%); and firms that use software as an input into their own operations rather than creating a software product, such as a retail firm that hires software engineers to maintain its website ("software as input", 13.6%).

We similarly classify each firm's customer industry, collapsing the industries into four categories. The largest single category is firms that build software serving a general business function rather than a specific industry, such as payroll management software (22.9%). Among firms serving a specific vertical, the most common are financial services and insurance (15.6%) and healthcare and life sciences (11.1%). The remaining firms (50.4%) span a range of other industries, each of which comprises a small share, including media, retail, education, and professional services.

The Jellyfish data covers a range of workers involved with the software production process, including but not limited to individuals who write code for software. We provide a description of the roles of workers in our analysis sample in Table 3. When available, we use job titles inputted by firms into Jellyfish. When that is not available, we use job titles obtained from the worker-level merge with LinkedIn. Software engineers make up the majority of workers (76.6%). The remainder

are split across data science and research (5.8%), other forms of engineering (e.g., hardware, 0.3%), product and design (7.9%), revenue (e.g., sales and marketing, 3.9%), general and administrative (e.g., legal and HR, 5.4%), and chief executives (0.2%).

2.3 Jira issue classification

Classification method. We measure final software output using the Jira data on issues and epics. Our analysis of changes in final output relies on classification of Jira issues and epics by their size. We assess the size of each Jira issue using two complementary methods.

First, we use a large language model (LLM) to label a sample of tasks, then scale those labels to the full dataset using a supervised learning approach. For the labeling step, we prompt the GPT-4o API to classify each issue based on the estimated hours an experienced software engineer would require to complete it, without the assistance of AI. The prompt provides the task summary, description, issue type, parent task summary, and project name as context, and we label a 0.5% sample of issues, stratified by company. To scale these labels to the full dataset, we use a supervised learning approach, which predicts this LLM-label from the text description of the Jira issue. To do this, we first convert each issue's text description into numerical embeddings, then train a supervised learning model to predict the GPT-assigned length label from the embeddings. Full details are provided in Appendix Section F.2.

Second, we construct a complementary measure of task size from Jira metadata using cycle time — the number of days elapsed between a task being opened and resolved. We train the same architecture described above to predict log cycle time from the embeddings, using issues resolved in 2022 as the training set (approximately 2 million issues with non-missing cycle time). We then apply this trained model to the entire dataset of Jira issues, to predict the amount of time it would take to resolve the issue.

Validation of supervised learning method. We validate our supervised learning approach in Appendix Table A2. We take the GPT-assigned labels as the true labels and the outputs of the supervised learning step as the predicted labels. The table displays a comparison of these labels for a holdout sample of tasks. Because the length metric is continuous, we use the mean squared error (MSE) as our main validation metric. We obtain an MSE of 0.410 (on a 0–3 scale), which indicates that our model performs reasonably well.

2.4 AI coding tools

Background to AI coding tools. Finally, we describe AI coding tools. We refer to three primary types of AI coding tools: AI coding assistants, AI coding agents, and AI code review tools. Appendix

Figure A3 displays examples of the visual interface observed by users when interacting with AI coding tools.

AI coding assistants are tools that generate real-time code suggestions within the software application in which developers write and edit code. As a developer types, the assistant predicts and autocompletes the next lines of code; developers can also describe a desired function in plain language and have the assistant generate an implementation. AI coding assistants first became available in business and enterprise license form in February 2023, with the release of GitHub Copilot's business and enterprise licenses. A wide range of AI coding assistants are now commercially available. We focus on GitHub Copilot and Cursor, two of the most widely adopted coding assistants, for which Jellyfish provides API integrations.

AI coding agents are tools that can semi-autonomously complete longer, multi-step tasks. Agents can accept a task description and semi-independently plan and execute a sequence of actions. AI coding agents first became available in November 2024, with the release of Cursor's AI agent tool. A wide range of AI coding agents are now commercially available; Claude Code is among the most widely adopted. Although AI coding assistants and agents differ in terms of their interfaces, agents' release largely represents advances in model capabilities that allow the tools to complete more complex tasks. In turn, we conceptualize these tools as providing similar types of shocks to firms, differing only in magnitude.

Finally, AI tools are also able to perform code reviews, i.e., analyze pull requests. They can read over code to flag potential bugs, security vulnerabilities, or other changes needed in the code, and help to generate comments.

All of these tools can generally be configured to use models from any of the leading AI laboratories, including OpenAI (GPT series), Anthropic (Claude series), and Google (Gemini series). Generally, new versions of AI models are available in each tool nearly immediately at release.

Adoption process. The firm-level technology procurement process typically involves several stages. It is often initiated by senior management. The implementation is generally led by the IT team, which manages enterprise technology usage. The IT team generally begins by evaluating different possible technology vendors — e.g., comparing GitHub Copilot with Cursor. Then, the team must get internal business approval and budget allocation. In larger organizations, business teams may also negotiate pricing and contract terms with the technology provider. The tool must then receive approval from legal and security teams, which assess issues related to data handling and privacy, intellectual property ownership, and compliance. Finally, some firms conduct a pilot program in which a small subset of engineers tests the tool before broader deployment (Rapoport et al., 2025; IBM, 2024).

We gather information on the procurement processes across different firms through qualitative interviews, and provide some illustrative examples in Appendix Section C. Two examples demonstrate the contrast between firms in the length of the process. In one firm, the Head of AI Analytics and Insights describes a structured process:

> *We've got four teams involved in purchasing any tech. One is on the sourcing side, and handles large contracts; they understand whether the contracts are financially favorable towards [the company]. You also have legal, who will have an opinion on whether [the company's] data and IP are protected. Security gets involved as well. Finally, we expect IT to own and operate the tech centrally. The process typically takes a few weeks minimum. That can also mean months. ChatGPT came out in November 2022. Fall of 2023 is when we initially acquired a few test licenses before we rolled it out.*

By contrast, another firm describes a more nimble and shorter process:

> *Our EA just asked IT and legal if it was ok to get it and got approval fast. Usually, this happens within a day for us.*

Typically, firms disallow usage of AI tools outside of their business license, due to concerns surrounding user inputs being used by AI companies for model training. In turn, firm-level adoption should serve as a genuine shock to AI usage at the firm. One Head of AI Analytics and Insights describes such concerns:

> *Our code is our intellectual property. What happens to the code that's exposed to GitHub Copilot? ... We said, 'do not use anything that is not approved by [the company].'*

Measurement. We use several data sources to measure firm- and individual-level adoption of AI tools. The details of this measurement process are described in Appendix Section F.1.

First, we have access to API data on several AI coding tools integrated with JF's platform — GitHub Copilot, Cursor, and Claude Code. The API records daily metadata on usage, including the active users, number of suggestions generated and accepted, and number of instances of chat activity.

Second, we detect usage of AI coding agents and AI code review tools from GitHub activity data. Specifically, we identify tool-specific bot accounts by matching GitHub user logins and display names against known patterns for AI tools (e.g., "copilot-swe-agent[bot]", "coderabbitai[bot]"), and additionally scan commit messages and pull request bodies for AI tool signatures (e.g., "Co-authored-by: Claude Code," "Created by Devin"). This approach provides broad coverage across AI agents — including Cursor Agent, Claude Code, Devin, GitHub Copilot Agent, and others — and code review tools, including CodeRabbit, Greptile, GitHub Copilot Reviewer, Cursor Bug Bot, Graphite, and Bito.

We combine these two sources to construct our three treatment variables. For *AI coding assistant* adoption, we use the earliest seat activation date measured across the GitHub Copilot and Cursor APIs. For *AI coding agent* adoption, we take the earliest signal across the Claude Code API and the

GitHub signals (bot account creation and commit/PR signatures). For *AI code review tool* adoption, we use the earliest GitHub signal.

Our measurement of AI tool take-up is incomplete in three ways. First, the API integrations for GitHub Copilot, Cursor, and Claude Code only measure take-up through a business or enterprise license — if individual software engineers use AI tools through an individual license, their usage would not be recorded in the JF data. In practice, firms generally disallow usage of AI tools through individual licenses, due to concerns about data and code security. Second, some firms may adopt AI tools without integrating data on those tools into JF's platform. This concern is mitigated by the fact that we can infer some tool usage from GitHub user logins and display names. Finally, the API for Cursor was only built by JF in January 2025 and Claude Code in June 2025. In turn, firms that adopted the tool prior to that In practice, the high rate of take-up we observe at the end of our sample window suggests that we do effectively capture most take-up of AI tools. Insofar as we are unable to perfectly observe take-up of AI tools,

Take-up rate. Figure 2 shows the rates of firm- and individual-level take-up of AI coding assistants and AI coding agents. Panel A shows that take-up of AI coding assistants (GitHub Copilot or Cursor) began earlier, climbing from near zero in February 2023 to 43.8% of firms by April 2024. We truncate the assistant take-up window at 16 months, both because we view this as the main period of AI assistant diffusion and to avoid overlap with the period of take-up of AI coding agents. In Panel B, the share of firms that have adopted any AI agent rises from near zero in October 2024 to 91.0% by January 2026 and 98.3% by July 2026, with the steepest acceleration occurring around the releases of Cursor Agent (November 2024) and Claude Code (February 2025), marked by vertical dashed lines.

Panels C and D display individual-level take-up of AI coding assistants and AI coding agents, respectively. We estimate each as an event study in which the treatment is the firm's adoption of the tool:

$$\text{Takeup}_{f,m} = \gamma_f + \delta_m + \sum_{k \neq -1} \pi_k \text{AI}^k_{f,m} + \varepsilon_{f,m}.$$

Here, $\text{Takeup}_{f,m}$ is the percent of the firm's baseline coders — workers with GitHub activity in the baseline window — who have personally taken up the tool by month m. We hold this set of workers fixed, so that the estimates capture changes in take-up rather than changes in who works at the firm. As in the main specification, $\text{AI}^k_{f,m} = \mathbb{1}\{m - \text{AI}_f = k\}$ indicates that month m is k months after the firm's adoption month AI_f, γ_f and δ_m are firm and month fixed effects, and we bin the periods beyond the event window and cluster standard errors by firm.

For both tools, take-up is close to zero in the months before adoption and rises sharply after. Take-up of AI assistants jumps in the month of adoption and continues to climb, reaching 37.58% of

baseline coders 16 months after adoption. Take-up of AI agents starts more slowly but rises steadily, reaching 35.30% 12 months after adoption. However, due to the limitations in measurement described above, both firm- and worker-level take-up rates should be read as lower bounds on true take-up. To the extent that we date firm adoption too late, this should generally attenuate our main estimates; to the extent that we understate individual take-up, it would instead overstate the per-worker effects in the IV specification.

AI spend. Finally, Panel E of Figure 2 displays monthly spend per worker on AI coding agents in July 2026, measured through the Cursor and Claude Code APIs — the two tools for which we observe billing data — for the minority of firms and workers for which this data is available. Two patterns stand out. First, spend is non-trivial for many workers. The median worker spends $178 per month, the mean worker spends $808, and 20% of workers spend over $1,000. The right tail is long: the 99th percentile is $9,007 per month. Second, spend is nonetheless small relative to the cost of labor. The median worker's spend is 2.58% of the monthly salary of an entry-level software engineer — $6,872, the 10th percentile of annual wages for software developers in the BLS Occupational Employment and Wage Statistics — and only 1.65% of workers spend more than that salary (U.S. Bureau of Labor Statistics, 2025).

3 Model

We begin by developing a model of firm production and AI adoption. The model formalizes the relationship between a technology-driven productivity shock, firm output, and labor demand. It most directly builds on two literatures: first, a literature on the effects of AI adoption, based on the importance of "bottlenecks" or "weak links" arising from complementarities between tasks (e.g., B. Jones, 2025; C. Jones & Tonetti, 2026); second, a literature from operations that characterizes how production systems respond when one stage speeds up, but another is unable to keep pace, or rework is needed (e.g., Little, 1961; Kingman, 1961; So & Tang, 1995).

In our model, production involves two inputs that are perfect complements: coding and review. Though AI adoption can increase productivity in coding, review can serve as a bottleneck if firms are unable to adjust inputs fully, and because adjustment inherently shifts labor away from coding. Furthermore, AI adoption can decrease the quality of code generated, decreasing review productivity and worsening the bottleneck. Both of these channels limit AI's effects on overall firm output and shape its effects on labor demand.

Although we write the model in terms of software production, the concepts apply to a range of production settings in which a firm uses multiple complementary inputs that are differentially

affected by AI, and for which AI may change the quality of an intermediate output. In particular, the setup of "implementation" versus "management" or "review" is ubiquitous across work domains. Though AI tools can speed up implementation processes, output is still limited by the speed of decision-making concerning what projects to pursue, or by the speed of verification; and the usage of AI tools can make these surrounding tasks more difficult as well.

Empirically, we estimate effects of two generations of AI tools — AI coding assistants and AI coding agents. Though some researchers have conceptualized assistants and agents as distinct technologies (e.g., Sarkar, 2025), we will conceive of them as creating different-sized shocks to coding productivity and the bug rate. In other words, we abstract away from differences in the cognitive experience associated with interacting with these iterations of tools, which have slight differences in interfaces. We view this as a reasonable simplification, since agentic tools arose largely from improvements in underlying model capabilities rather than from a new technology (e.g., Kwa et al., 2025).

3.1 Set-up

Firm production. A firm ships software output. Production is sequential and involves two steps: coding and review. First, coders write draft code, and the quantity produced in period t is $C_t^P = A_{C,t}L_{C,t}$, where $A_{C,t}$ is labor productivity in coding and $L_{C,t}$ is the amount of coding labor. Second, reviewers assess code for bugs before approving it. Reviewers detect bugs with probability π_t. If the reviewer finds a bug, they send it back to the coder for revision. Thus, the quantity of code reviewed is $R_t^P = A_{R,t}L_{R,t}$ — where the effective review productivity is the raw review productivity, net of the bug rate, $A_{R,t} := (1-\pi_t)\tilde{A}_R$, and the amount of review labor is $L_{R,t}$. The code and reviews that are put into production — i.e., the amount that is actually used by the firm to generate revenue — are C_t, R_t. Some code and reviews that are produced do not actually have to be put into production — e.g., if code is drafted but unreviewed. This difference allows for the emergence of a bottleneck:

$$\int_{\tau=0}^{t} C_\tau \, \mathrm{d}\tau \;\leq\; \int_{\tau=0}^{t} C_\tau^P \, \mathrm{d}\tau \qquad \int_{\tau=0}^{t} R_\tau \, \mathrm{d}\tau \;\leq\; \int_{\tau=0}^{t} R_\tau^P \, \mathrm{d}\tau.$$

Finally, each unit of software output bundles together n approved code changes — e.g., an update to a software feature may require a user interface code change as well as a back-end code change. This variable may also change in response to AI adoption — for instance, workers may submit code for review more frequently. Taken together, the total amount of software output produced by the firm is Leontief in the code and reviews put into production, and scaled by the units of approved code

needed for a unit of software output,

$$Y_t = \frac{1}{n} \min\{C_t, R_t\}.$$

Workers. The firm employs L total workers, paid wage w, of whom a share ϕ are senior. In the short and medium run, total labor is fixed and normalized to $L = 1$. The firm has the ability to allocate workers between code and review. Workers are largely substitutable between these two stages. However, a share of review work ρ may only be completed by a senior worker. Thus, review labor will be bounded.

$$\rho\, L_{R,t} \leq \phi, \qquad \text{that is} \qquad L_{R,t} \leq \bar{L}_R := \frac{\phi}{\rho}.$$

We can think of ρ as being a feature of the code. For instance, if a given code change is more complex, or covers a product that is more consequential — such as an infrastructure change that would affect many parts of the software product; or a product that has a large client base — then the firm may employ stricter review standards. These assumptions are consistent with modern practices for code review, which we can validate both qualitatively and empirically. For instance, one software engineer describes the typical assignment process:

> *When teams assign code review, assignment usually boils down to availability. But, if a PR or area of code is complex, I expect at least one reviewer to be senior.*

Time to approval. The time between draft code submission and code approval will consist of two lengths: the length of time that the code is waiting to be reviewed, in a backlog, and the length of time that the code is in review.

The size of the backlog is the difference between the code written and the code reviewed. Code will accumulate at the review stage if coding capacity outpaces review capacity. Per Little's Law (Little, 1961), the backlog is cleared at the rate of review capacity, so the wait time is the quotient of the backlog size and review capacity.

$$B_t = \int_0^t \left[C_\tau^P - R_\tau^P\right] \mathrm{d}\tau \;\geq\; 0, \qquad W_t = \frac{B_t}{R_t^P}.$$

The length of the review process itself will be the inverse of the review productivity — treating each code change as reviewed by one reviewer from start to finish, so that the reviewer time it requires is also its elapsed time in review. This is a function of the raw review productivity and the bug rate:

$$\frac{1}{A_{R,t}} = \frac{1}{\tilde{A}_R} \cdot \frac{1}{1 - \pi_t}.$$

To interpret this expression, note that $\frac{1}{\bar{A}_R}$ is the amount of reviewer time required for one review round. Then, let $X_i = 1$ be an indicator that review round i does not contain a bug, such that $X_i \sim \text{Bern}(1-\pi_t)$, independently across rounds. The number of rounds needed for approval is then the trial of first success, $N \sim \text{Geom}(1-\pi_t)$ with mean $\mathbb{E}[N] = \frac{1}{1-\pi_t}$.

Putting these two lengths together, the overall time from draft code submission to approval is:

$$\underbrace{W_t}_{\text{waiting for a reviewer}} + \underbrace{\frac{1}{A_{R,t}}}_{\text{in review}}.$$

3.2 The firm's problem

The firm discounts at rate r and chooses how to allocate labor between the two production stages to maximize the present value of profit, taking the price of software output P as given. In the short and medium run, total labor is fixed and normalized to one, so the wage bill is a constant, which we describe in this section and Section 3.3. In the long run, the firm also chooses how much labor to employ, which we describe in Section 3.4.

$$\max_{\{L_{C,t}, L_{R,t}, C_t, R_t\}_t} \int_0^\infty e^{-rt}\Big[P\, Y_t - w\big[L_{C,t} + L_{R,t}\big]\Big]\,\mathrm{d}t$$

subject to (1) Final output: $Y_t = \dfrac{1}{n}\min\{C_t, R_t\}$,

(2) Intermediate outputs: $C_t^P = A_{C,t}L_{C,t}$ and $R_t^P = A_{R,t}L_{R,t}$,

(3) Deployment: $\displaystyle\int_{\tau=0}^{t} C_\tau\,\mathrm{d}\tau \le \int_{\tau=0}^{t} C_\tau^P\,\mathrm{d}\tau$ and $\displaystyle\int_{\tau=0}^{t} R_\tau\,\mathrm{d}\tau \le \int_{\tau=0}^{t} R_\tau^P\,\mathrm{d}\tau$,

(4) Total labor: $L_{C,t} + L_{R,t} = 1$,

(5) Review ceiling: $L_{R,t} \le \bar{L}_R$,

(6) Backlog: $B_t = \displaystyle\int_{\tau=0}^{t}\big[C_\tau^P - R_\tau^P\big]\,\mathrm{d}\tau \;\ge\; 0.$

We solve the firm's optimization problem to obtain its pre-AI labor allocation, output, and time to review approval.

Proposition 1 (The firm's optimum)**.** *Suppose the review ceiling is slack, $\bar{L}_R \ge A_C/[A_C + A_R]$. Then the firm's output, labor allocation, and review backlog are characterized by the following expressions:*

1. *Labor is allocated as follows:*

$$L_C^* = \frac{A_R}{A_C + A_R}, \qquad L_R^* = \frac{A_C}{A_C + A_R}.$$

2. *Intermediate and final outputs are:*

$$C^{P*} = C^* = R^* = R^{P*} = \frac{A_C A_R}{A_C + A_R}, \qquad Y^* = \frac{C^*}{n}.$$

3. *There is no backlog or wait time, $B_t = 0$ and $W_t = 0$. The time from submission to approval is the review time, $1/A_R$, alone.*

We leave the full proof, as well as those that follow, to Appendix G. In short, with Leontief production, the firm will allocate labor to balance coding and review capacities $C^P = R^P \iff A_C L_C = A_R L_R$. Because the backlog does not enter the objective, holding code back is never profitable, and with time discounting it is strictly costly: the firm puts all code and reviews into production, $C^P = C$ and $R^P = R$. In turn, the dynamic optimization problem collapses to a static one within each time period. We therefore drop the time subscript on technology, labor, and flows from here on, and retain it only on the backlog B_t and the waiting time W_t, which are still time-dependent.

Two features matter for the firm's response to AI adoption below. First, L_C^* and L_R^*, which are the optimal pre-AI allocations of labor, are also each stage's share of the worker-time a unit of output requires. These relative shares will shape the extent to which a productivity change in each stage passes through to output. Second, the labor allocation is shaped by the relative productivities A_C/A_R: e.g., a firm whose coders are more productive, whose reviewers are relatively less productive, or whose code is buggier, will allocate more labor towards review; and vice versa.

3.3 Effects of AI adoption, short and medium-run

In this section, we characterize the impacts of firm-level AI adoption. AI can affect firm production by increasing coding productivity A_C or by increasing the bug rate π. We begin by characterizing the impacts of these two channels individually. Then, we characterize the firm's response and the impacts on output and employment across three time horizons: the short run, with no labor reallocation; the medium run, with internal labor reallocation; and the long run, with fully adjustable labor. Throughout this section, elasticities are evaluated at the optimum of Proposition 1.

Lemma 1 (Coding productivity channel)**.** *An increase in A_C, holding π and the allocation fixed, raises coding capacity and leaves review capacity and time in review untouched:*

$$\frac{\partial \ln C^P}{\partial \ln A_C} = 1, \qquad \frac{\partial \ln R^P}{\partial \ln A_C} = 0, \qquad \frac{\partial \ln[1/A_R]}{\partial \ln A_C} = 0.$$

The backlog grows at $\dot{B} = C^P - R^P > 0$ and the waiting time at $\dot{W} = \dot{B}/R^P$. The increase in the time to approval, $W_t + 1/A_R$, comes entirely from an increase in the waiting time.

Lemma 2 (Bug rate channel)**.** *An increase in π, holding A_C and the allocation fixed, lowers effective review productivity and thus lengthens time in review:*

$$\frac{\partial \ln C^P}{\partial \ln \pi} = 0, \qquad \frac{\partial \ln A_R}{\partial \ln \pi} = \frac{\partial \ln R^P}{\partial \ln \pi} = -\frac{\pi}{1-\pi} < 0, \qquad \frac{\partial \ln[1/A_R]}{\partial \ln \pi} = \frac{\pi}{1-\pi} > 0.$$

The backlog grows at $\dot{B} = C^P - R^P > 0$ and the waiting time at $\dot{W} = \dot{B}/R^P$. Both terms in the time to approval, $W_t + 1/A_R$, rise.

These lemmas describe the two ways in which AI adoption can affect firm production. In the short run, both channels generate a bottleneck in the review stage, though through different means. An increase in coding productivity generates a bottleneck because coding capacity outpaces review capacity. In turn, the waiting time at the review stage will increase, but the review time itself will not change. An increase in the bug rate generates a bottleneck through a spillover to the review stage, i.e., a decrease in effective review productivity. In turn, the review time itself will increase, and the waiting time will as well.

We next characterize the impacts of these two channels on firm output and labor demand, under different time horizons. We begin by characterizing a short-run effect, in which the firm has not adjusted any of its labor inputs.

Proposition 2 (Short run, with no labor reallocation)**.** *Suppose the firm holds labor allocation at its pre-AI level. Once the firm adopts AI, increasing A_C and π, coding capacity exceeds review capacity, $C^P > R^P$, and:*

1. *Output is constrained by the review capacity:*

$$Y = \frac{R^P}{n}.$$

2. *A coding productivity gain does not pass through to output at all; a higher bug rate lowers output:*

$$\frac{\partial \ln Y}{\partial \ln A_C} = 0, \qquad \frac{\partial \ln Y}{\partial \ln \pi} = -\frac{\pi}{1-\pi}.$$

3. *The backlog and wait time both grow:*

$$\dot{B} = C^P - R^P > 0, \qquad W_t = \frac{B_t}{R^P}.$$

In the short run, because production is Leontief, if the firm has not reallocated labor at all, a significant coding productivity gain has no effect on overall output, because review serves as a binding constraint. If AI also increases the bug rate, then its overall effect on output is negative, since it also decreases review productivity. Through either of these channels, coding capacity will outpace review capacity,

which will generate a backlog and an increase in waiting time at the review stage.

Next, we characterize a medium-run effect, in which the firm's overall employment is held fixed, but the firm can reallocate inputs internally. Write $L_R^{*\prime}$ for the review labor that would equalize the two stages at the new technology. We will separately characterize the effects of AI adoption, if the review ceiling does versus does not bind.

Proposition 3 (Medium run, review ceiling slack)**.** *Suppose overall labor L is held fixed, but the internal labor allocation is not held fixed, and the review ceiling does not bind, $L_R^{*\prime} \leq \bar{L}_R$. Once the firm adopts AI, increasing A_C and π, then:*

1. *Each channel passes through to output with its own weight:*

$$\frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln A_C} = L_C^* \;>\; 0, \qquad \frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln \pi} = -L_R^* \frac{\pi}{1-\pi} \;<\; 0.$$

2. *Labor moves toward review under both channels, and the relative labor allocation is:*

$$\frac{L_R^*}{L_C^*} = \frac{A_C}{A_R}, \qquad \frac{\mathrm{d}\ln[L_R^*/L_C^*]}{\mathrm{d}\ln A_C} = 1, \qquad \frac{\mathrm{d}\ln[L_R^*/L_C^*]}{\mathrm{d}\ln \pi} = \frac{\pi}{1-\pi}.$$

3. *The backlog and wait both stop growing: $\dot{B}^* = 0$ and $\dot{W}^* = 0$.*

Proposition 4 (Medium run, review ceiling binds)**.** *Suppose instead that $L_R^{*\prime} > \bar{L}_R$, i.e., the review ceiling binds. Once the firm adopts AI, then:*

1. *Output is capped by review capacity at the review ceiling. Additional coding productivity gains do not pass through to output, and increases in the bug rate further decrease output:*

$$Y^* = \frac{A_R \bar{L}_R}{n}, \qquad \frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln A_C} = 0, \qquad \frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln \pi} = -\frac{\pi}{1-\pi}.$$

2. *Labor stops responding, to either channel:*

$$L_R = \bar{L}_R, \qquad L_C = 1 - \bar{L}_R, \qquad \frac{\mathrm{d}\ln L_j}{\mathrm{d}\ln A_C} = \frac{\mathrm{d}\ln L_j}{\mathrm{d}\ln \pi} = 0, \qquad j \in \{C, R\}.$$

3. *The queue keeps growing: $\dot{B}^* = A_C[1 - \bar{L}_R] - A_R \bar{L}_R \;>\; 0$.*

Both the productivity and bug rate channels push the firm to reallocate labor towards review. In the medium run, the firm is able to perform this reallocation until reaching a review ceiling $\bar{L}_R$. Thus, the pass-through of the coding productivity shock can be limited in two ways. First, even if the firm is able to adjust perfectly — i.e., the review ceiling does not bind — the coding gain still raises output only in proportion to L_C^*, because the firm must reallocate labor towards review. In turn, the effect of AI adoption on final software output is still only a fraction of the effect on intermediate code output. Second, if the review ceiling binds and the firm is not able to reallocate completely, then the backlog and wait time will continue to grow, and pass-through will be lower.

3.4 Effects of AI adoption, long-run adjustment

Finally, we turn to considering the firm's adjustment in the long run, in which total labor is no longer fixed. Instead, the firm chooses overall labor L as well as the relative allocations L_C, L_R freely. With total labor free, the review ceiling applies to review's share of labor: the senior share ϕ is unchanged, so $\rho L_R \leq \phi L$, and review's share of labor, L_R/L, cannot exceed $\bar{L}_R = \phi/\rho$. Additionally, scale must be pinned down by demand rather than by the labor endowment. We take demand to be linear, as is common in models of imperfect competition in which the demand elasticity varies along the demand curve (Melitz & Ottaviano, 2008; Weyl & Fabinger, 2013; Mrázová & Neary, 2017),

$$P_t = a - b\ Y_t,$$

and write ε for the elasticity of demand evaluated at the firm's pre-adoption optimum, $\varepsilon := -\mathrm{d}\ln Y/\mathrm{d}\ln P = P/[a-P]$. As the monopolist always prices on the elastic portion of the demand function, $\varepsilon > 1$.

The firm's problem is then the one in Section 3.2, with the labor constraint $L_{C,t} + L_{R,t} = 1$ replaced by the product demand curve and the price now endogenous.

$$\begin{aligned}
\max_{\{L_{C,t},\, L_{R,t},\, C_t,\, R_t,\, Y_t\}_t} \quad & \int_0^\infty e^{-rt}\Big[P_t\, Y_t - w\big[L_{C,t} + L_{R,t}\big]\Big]\,\mathrm{d}t \\
\text{subject to} \quad & (1)\ \ Y_t = \frac{1}{n}\min\{C_t, R_t\}, \\
& (2)\ \ C_t^P = A_{C,t}L_{C,t} \ \text{ and } \ R_t^P = A_{R,t}L_{R,t}, \\
& (3)\ \ \int_{\tau=0}^{t} C_\tau\,\mathrm{d}\tau \leq \int_{\tau=0}^{t} C_\tau^P\,\mathrm{d}\tau \ \text{ and } \ \int_{\tau=0}^{t} R_\tau\,\mathrm{d}\tau \leq \int_{\tau=0}^{t} R_\tau^P\,\mathrm{d}\tau, \\
& (4)\ \ P_t = a - b\ Y_t, \\
& (5)\ \ \rho\, L_{R,t} \leq \phi\,\big[L_{C,t} + L_{R,t}\big], \\
& (6)\ \ B_t = \int_{\tau=0}^{t}\big[C_\tau^P - R_\tau^P\big]\,\mathrm{d}\tau \ \geq\ 0.
\end{aligned}$$

With this set-up, we can solve for the firm's optimum, with the scale of output and overall labor determined by product demand.

Lemma 3 (The firm's optimum with product demand)**.** *Suppose the review ceiling is slack, $A_C/[A_C + A_R] \leq \bar{L}_R$. Then the firm's optimal price, output, and labor allocation are characterized by the following expressions:*

1. *The wage bill per unit of output, c, and price markup are the following:*

$$c = w\,n\,\frac{A_C + A_R}{A_C A_R}, \qquad P = \frac{\varepsilon}{\varepsilon - 1}\,c.$$

2. *Demand then pins down output, which determines the labor choices:*

$$Y = \frac{a - c}{2b}, \qquad L_C = \frac{nY}{A_C}, \qquad L_R = \frac{nY}{A_R}, \qquad L = \frac{cY}{w}.$$

3. *All code produced is put into production, and code and review capacities are equalized, so* $B_t = 0$ *and* $W_t = 0$.

As before, we drop the time subscripts from here on, as the problem reduces to a static one within each time period. Now, demand determines the scale of output and overall labor. However, as before, the relative labor allocation is determined by relative productivities: the shares L_C/L and L_R/L equal L_C^* and L_R^* from Proposition 1. With this set-up, we can characterize the firm's long-run adjustment in response to AI adoption, with flexible output and labor.

Proposition 5 (Long run, with employment adjustment)**.** *Once the firm adopts AI, increasing* A_C *and* π*, the outcomes respond as follows:*

1. *Output rises with coding productivity and falls with the bug rate, scaled by the elasticity of product demand:*

$$\frac{\mathrm{d}\ln Y}{\mathrm{d}\ln A_C} = \frac{\varepsilon - 1}{2} L_C^* \;>\; 0, \qquad\qquad \frac{\mathrm{d}\ln Y}{\mathrm{d}\ln \pi} = -\frac{\varepsilon - 1}{2} L_R^* \frac{\pi}{1 - \pi} \;<\; 0.$$

2. *Overall labor responds as follows, with ambiguous sign:*

$$\frac{\mathrm{d}\ln L}{\mathrm{d}\ln A_C} = \left[\frac{\varepsilon - 1}{2} - 1\right] L_C^*, \qquad\qquad \frac{\mathrm{d}\ln L}{\mathrm{d}\ln \pi} = -\left[\frac{\varepsilon - 1}{2} - 1\right] L_R^* \frac{\pi}{1 - \pi}$$

3. *Coding and review labor respond as follows:*

$$\frac{\mathrm{d}\ln L_C}{\mathrm{d}\ln A_C} = \frac{\varepsilon - 1}{2} L_C^* - 1, \qquad \frac{\mathrm{d}\ln L_C}{\mathrm{d}\ln \pi} = -\frac{\varepsilon - 1}{2} L_R^* \frac{\pi}{1 - \pi} < 0,$$
$$\frac{\mathrm{d}\ln L_R}{\mathrm{d}\ln A_C} = \frac{\varepsilon - 1}{2} L_C^* > 0, \qquad \frac{\mathrm{d}\ln L_R}{\mathrm{d}\ln \pi} = \left[1 - \frac{\varepsilon - 1}{2} L_R^*\right] \frac{\pi}{1 - \pi}.$$

There are several features that are notable about the firm's long-run decisions. First, an increase in coding productivity A_C unambiguously raises output and an increase in the bug rate π unambiguously decreases it. However, the magnitudes of these effects will depend on the elasticity of product demand ε and baseline labor allocation L_C^*, L_R^*. For instance, if product demand is more elastic or if coding is a larger share of production at baseline, the impact of a coding productivity shock will be larger.

Second, the signs of the effects of both channels on overall labor are ambiguous. To see this, note that the comparative statics in item 2 can be characterized as threshold conditions — for instance,

above the threshold $\varepsilon > 3$, the coding productivity shock will lead to a net increase in employment, as the scale effect of increased output will dominate the displacement effect of needing less labor per unit of output. If demand is less elastic, the reverse will be true. The bug rate's effect on employment turns on the same threshold, with the opposite sign.

One can also separately characterize the firm's demand for the two forms of labor. An increase in coding productivity unambiguously raises the amount of review labor demanded, as more code will need to be reviewed. However, it will have ambiguous effects on the amount of coding labor, due to the opposing scale and displacement effects. With similar logic, an increase in the bug rate has ambiguous effects on the amount of review labor demanded. However, it will unambiguously decrease the amount of coding labor demanded, as output will fall but the amount of coding labor per unit of output does not change.

3.5 Testable implications

The model yields several testable implications for the short- to medium-run impacts of AI adoption. In Sections 4 and 5, we provide reduced form evidence on these outcomes.

1. *Review ceiling.* Senior review capacity caps how far output can grow. We measure this ceiling as the share of engineers who are senior divided by the share of code that requires senior review.
2. *Coding productivity.* If AI raises coding productivity, code output should rise. We measure code output as lines of code added, commits, and pull requests opened.
3. *Final output and labor adjustment.* If AI raises coding productivity, and any labor has been reallocated to review, then final software output will rise. We measure final software output as the number of Jira issues resolved, indicating deployment of the associated code to production; and the number of Jira epics resolved, indicating completion of larger software projects.
4. *Bug rate.* If AI raises the bug rate, then code will spend longer in review. We measure the bug rate as the share of pull requests that receive a comment or are blocked from being merged. We measure the length of review as the time between the first comment on a pull request and the time the pull request is merged.
5. *Backlog and wait times.* If AI affects either coding productivity or the bug rate, the review backlog and wait times should grow. They keep growing unless the firm optimally reallocates labor toward review. We measure the wait time as the time from the pull request being opened to the first comment. We measure the backlog size as the number of pull requests that are unreviewed (i.e., unmerged and have not received a comment).

Then, in Section 6, we estimate the model and use it to provide evidence on the projected long-run

effects of AI adoption, per Lemma 3 and Proposition 5.

4 Effects on code output, software output, and employment

In this section, we document the effects of AI adoption on intermediate code output, final software output, and employment.

4.1 Descriptive evidence

Before turning to causal estimates, we describe the raw data on the paper's main outcomes.

Appendix Figure A4 plots the distribution of code output across firms in three groups: adopters before adoption, adopters after adoption, and firms that never adopt. We measure code output in three ways: (1) lines of code added; (2) GitHub commits, which are incremental updates to code; and (3) GitHub pull requests created, which are completed drafts of code submitted for review. We display the firm-level distributions of these outcomes, normalized by the number of workers in the firm, pre-AI adoption, post-AI adoption, and among non-adopters.

Appendix Figure A5 shows the same distributions for our two measures of final software output. As described in Section 2, software engineering teams use Jira to track work, which gives two natural measures of output. (1) The number of Jira *issues* resolved per worker per month. Issues are discrete units of engineering work — such as a bug fix or feature update — and are marked as complete when the associated code is deployed to production. (2) The number of Jira *epics* resolved per worker per month. Epics are larger software projects or features — i.e., groups of software tasks. Appendix Figure A6 shows the distributions for two measures of code review: the number of days from when a pull request is opened to when it is merged, and the number of comments per pull request.

These distributions preview the paper's main findings. Code output per worker is clearly higher after adoption. For lines of code added, the median firm writes 3,115 lines per worker per month before adopting AI assistants and 4,424 after, a 42% increase. The gap is larger for AI agents: the median rises from 3,679 to 6,482 lines, or 76%. Commits and pull requests show the same pattern on a smaller scale. The median number of commits rises 14% after AI assistant adoption (26.47 to 30.15) and 23% after agent adoption (27.94 to 34.45). The median number of pull requests created rises 15% (4.84 to 5.59) and 32% (4.97 to 6.55). Each of these outcomes has a long right tail, and the tail also shifts out after adoption: for lines of code, the 90th percentile rises from 10,609 to 14,753 for AI assistants and from 14,750 to 18,580 for AI agents.

Final software output also rises, but by less. For Jira issues resolved, the median rises 9% after AI assistant adoption (2.53 to 2.76) and 11% after agent adoption (2.76 to 3.08). Jira epics barely move: the median goes from 0.125 to 0.137 per worker per month for AI assistants, and from 0.137 to 0.157

for AI agents. Finally, the code review distributions point to the mechanism. After agent adoption, the median firm's pull requests receive 1.89 comments, compared with 1.21 before, a 57% increase. There is no such change after AI assistant adoption (1.49 to 1.44).

These are raw differences, however, and two confounds could drive them. First, adopters may differ in level from firms that never adopt: for example, firms that adopt AI tools may be more productive to begin with. Second, outcomes trend over time: for example, firms grow. The difference-in-differences design below addresses both.

4.2 Methodology

Empirical specification. To assess the impacts of AI coding assistant and AI coding agent adoption, we use a difference-in-differences design, exploiting variation in the timing of firm-level adoption of assistants and agents. We implement the following regression for firms f and months m:

$$\text{Activity}_{f,m} = \gamma_f + \delta_m + \sum_{k \neq -1} \beta_k \text{AI}^k_{f,m} + \theta_m X_{f,m} + \varepsilon_{f,m}.$$

Here, $\text{Activity}_{f,m}$ represents our outcome variables, which are measured at the firm-by-month level, such as the number of Jira issues resolved per worker. Our main variable of interest is $\text{AI}^k_{f,m} = \mathbb{1}\{m - \text{AI}_f = k\}$, where AI_f is the first month of adoption, so $\text{AI}^k_{f,m}$ is an indicator for whether the firm has adopted in relative month k.

We include firm fixed effects γ_f to control for time-invariant firm characteristics that may confound our estimates, such as baseline productivity or industry, as well as month fixed effects δ_m to control for time-varying characteristics, such as macroeconomic conditions. We also include time-varying controls for baseline firm size $X_{f,m}$, motivated by imbalance in treatment take-up, as described below. One feature of our data is that firms onboard gradually relative to their timing of joining Jellyfish; in turn, we observe a gradual increase in data coverage. Thus, we include a control for months since onboarding onto Jellyfish. For each tool, the sample is the firms that adopt it during our sample period: effects are identified from differences in the timing of adoption, with firms that have not yet adopted serving as the comparison group for firms that have. Finally, we cluster standard errors at the firm level, since treatment is defined at the firm level. We bin the pre- and post-periods beyond the main event study window into one indicator, to improve the precision of our estimates. We examine robustness to several alternative ways of estimating the effects of AI adoption in Section 4.4.

We separately estimate these effects for adoption of AI coding assistants and AI coding agents. In turn, the estimated effect of AI coding assistants represents the effects of initial AI adoption. By contrast, the estimated effect of AI coding agents represents the added effect of agent adoption,

relative to AI assistant adoption. We estimate the effects across 8 pre-AI adoption periods for both treatments. For AI assistants, we estimate effects across 16 post-adoption months; we view this as the reasonable timeframe with plausibly exogenous variation in adoption timing for AI assistants, based on the timing of take-up and our qualitative conversations regarding the procurement process; we cut it off at 16 months to avoid overlapping months with the AI agent figures. For AI agents, we estimate effects across 12 months of post-period, as the technology is newer, so few firms have more than 12 post-periods of data available, and our estimates become noisy past that time. For comparability across the two tools, the estimates we report in the text are the coefficients twelve months after adoption.

We are primarily interested in the firm-level effect of access to generative AI tools, rather than an individual-level effect, for two reasons. First, we are interested in the effects of AI adoption on firm-level outcomes, including output and employment. In turn, although the effect on coding productivity can be measured at an individual level, we are primarily interested in the aggregated firm-level estimate, though we do also report the effects from an instrumented version of our specification in Section 4.4. Second, the firm-level treatment represents the economically relevant margin for firms making decisions about whether to adopt generative AI tools.

Identification assumptions. Our identification strategy relies on the assumption that, absent treatment, earlier- and later-adopting firms would have followed parallel trajectories in outcomes. The technology adoption process is primarily shaped by two factors: (1) the timing by which firms learn about AI tools, and (2) the length of firms' AI procurement processes. We detail this process in Section 2.4 and Appendix Section C (Rapoport et al., 2025; IBM, 2024).

We may be concerned that some factors shape both the timing of adoption and firms' outcome trajectories. We assess this possibility by first checking for balance on observable characteristics in Table 4. Here, we observe that firm size — as measured by overall employment in Revelio as well as engineering employment in Jellyfish — is predictive of take-up timing for both AI coding assistants and AI coding agents, with larger firms adopting AI earlier. Other observable factors, including firms' product type and industry, do not exhibit consistent relationships with take-up timing. This relationship is consistent with existing empirical evidence on adoption of AI (Yotzov et al., 2026; Chatterji et al., 2026), as well as with qualitative accounts of the technology adoption process, in that larger firms are generally more knowledgeable about frontier technologies than smaller firms. This pattern would pose issues for identification if we also expect that these larger firms would exhibit different outcome trajectories — for instance, if they are also growing more quickly. The imbalance in take-up motivates our inclusion of time-varying controls for baseline firm size in our main specification.

Beyond firm size, procurement typically involves a number of logistical factors that are plausibly orthogonal to firms' growth trajectories, but shape the timing of adoption, as described in Section 2.4 and Appendix Section C. Firms must go through a number of approval layers — business, legal, security — and often will pilot tools with a small group of workers prior to purchasing a business license. The lengths of these processes vary significantly across firms, generating quasi-random variation in timing of adoption, which we leverage for our identification strategy.

We assess the parallel trends assumption empirically by examining pre-period coefficients, displayed in the results of the sections below. To the extent that unobserved confounders remain, we would generally expect them to produce upward bias: factors that accelerate adoption would also tend to produce higher outcome growth, leading us to overstate treatment effects. However, we largely find that the effects of AI are quite *small*, including that they are *smaller* than those found by others who study the impacts of AI on software engineering. Furthermore, insofar as any endogeneity concerns should produce bias in our results, the bias should show up similarly across our event studies. The focus of the paper is largely on the pass-through of the estimates of effects on coding output to final software output and employment — and we do not have reason to believe that the endogeneity concerns should differ across these outcomes.

4.3 Effects on intermediate code output

First, we document the effects of AI adoption on code output. As described before, we examine three measures of code output: (1) lines of code added, (2) commits, and (3) pull requests opened.

Figure 3 displays our estimated effects on code output. Panels A and B display event studies for the estimated effects of AI assistants and AI agents on lines of code added per worker-month. For AI assistants, twelve months after adoption, the effect is 1,045.9 lines per worker-month, or 30.4% of the baseline mean of 3,445.3, though this estimate is marginally insignificant. For AI agents, twelve months after adoption, the effect is 2,113.6 lines per worker-month, or 44.0% of the baseline mean of 4,806.1, and this estimate is also marginally insignificant.

One concern is that the change in lines of code reflects the verbosity of AI-generated code rather than a genuine increase in coding output. This would both inflate the measured code output effect and introduce additional noise into our estimates. To assess this, we also examine the effects on the number of commits per worker-month in Panels C and D and pull requests per worker-month in Panels E and F. For commits, the effect for AI assistants is positive and significant: 3.11 commits per worker-month, or 16.8% of the baseline mean of 18.55. For AI agents, we also find a significant effect of 9.51 commits per worker-month, or 36.0% of the baseline mean of 26.43.

Pull requests represent a coarser unit of work than commits, representing larger code changes that must be "opened" for review, and then "merged" once review is completed. The effect of AI

assistants on PRs opened is moderately large and significant: 0.734 pull requests opened per worker-month, or 20.9% of the baseline mean of 3.52. For AI agents, we find a large and significant effect of 1.73 pull requests opened per worker-month, or 35.7% of the baseline mean of 4.83.

Finally, after being reviewed, pull requests get merged. The number of pull requests merged per worker reflects not only the rate at which code enters review — i.e., coding productivity and labor — but also the rate at which it leaves review — i.e., review productivity and labor. The results are displayed in Appendix Figure A13. For AI assistants, the increase in merges is close to the increase in pull requests opened: 0.581 additional pull requests merged per worker-month, or 19.4% of the baseline mean of 3.00, compared to 20.9% for pull requests opened — roughly 79.2% of the additional pull requests opened are merged, in line with the baseline rate of 85.4%. For AI agents, by contrast, we observe a significant increase, though not as large as on pull requests opened: 1.08 additional pull requests merged per worker-month, or 26.0% of the baseline mean of 4.16, compared to 35.7% for pull requests opened. Put differently, only about 62.7% of the additional pull requests opened following agent adoption are merged, well below the 86.1% of pull requests merged at baseline.

As a validation of our methodology, Appendix Figure A9 benchmarks our estimates of the productivity effects of AI coding assistants against those reported in the broader literature (Cui et al., 2024; Hoffmann et al., 2024; Song et al., 2024; Sarkar, 2025; Demirer, Musolff, & Yang, 2026). Our estimates are broadly comparable in magnitude, lending confidence that the results are not driven by features specific to our sample or design.

Anecdotal accounts of software engineers' experiences with AI also support these empirical findings. For instance, one senior software engineer describes how AI tools have significantly sped up the process of writing code, particularly on simpler coding tasks:

> *For simple and routine tasks, there's so much documentation, so the models are very well-trained. With Claude Code, I can write a draft in two seconds and it's going to be right 90% of the time; I just have to do a little bit of testing myself. There is a huge speed-up, on the order of 100x.*

4.4 Robustness and alternative specifications

We test the robustness of our results to alternative estimators and specifications. Appendix Figure A7 displays the results, focusing on the effects of agent adoption on lines of code added.

First, a recent literature shows that two-way fixed effects estimates can be biased when adoption is staggered and treatment effects vary across firms or over time (Borusyak et al., 2024; de Chaisemartin & D'Haultfœuille, 2024; Callaway & Sant'Anna, 2021; Sun & Abraham, 2021). Panel A compares our two-way fixed effects estimates with the estimators proposed in these papers. The estimates are similar across estimators: all show flat pre-periods and an increase in lines of code, with similar magnitudes of effects and time patterns. This matches a recent survey of published papers, which

finds that heterogeneity-robust estimators rarely overturn the results of standard two-way fixed effects (Hull, 2026). We therefore use two-way fixed effects for the rest of the paper.

Second, Panels B–E vary the specification. Twelve months after adoption, the main specification gives an effect of 44.0% of the baseline mean. Adding the firms that never adopt agents to the comparison group (Panel B) gives 57.7%. Dropping the controls for baseline firm size and for the start of each firm's data coverage (Panel C) gives 38.3%. Panel D restricts the sample of Panel B to a balanced panel of adopters: firms observed in every month from 8 months before to 12 months after adoption, so the same firms lie behind every event-time coefficient. This gives 73.8%, with no evidence of pre-trends ($p = 0.624$). We keep the never adopters as controls here because, without them, the balanced panel has almost no firms left to serve as controls in the last months of the data: by then, every adopter observed for 12 months after adoption has already adopted. Finally, measuring lines of code as a firm-level total rather than per worker (Panel E) gives 77.6%. In each case the pre-period coefficients remain close to zero, and the effect is similar to or larger than in the main specification, except when we drop the controls, where it is somewhat smaller.

Finally, we also estimate an IV specification, with results displayed in Appendix Figure A8. Our main specification estimates the effect of firm-level AI adoption: the change in the outcome when the firm adopts the tool, regardless of whether or not each worker uses it. However, it understates the effect on a worker who uses the tool, because many workers do not take it up. Furthermore, the dynamics of the event studies conflate changes in person-level take-up with other dynamics associated with AI take-up, including learning how to use the tools and improvements in model capabilities. To recover the effect per worker who uses the tool and shed light on these different time dynamics, we estimate the effect of worker adoption of AI, by using firm adoption as an instrument for individual take-up. We estimate the following system by two-stage least squares:

$$\text{Activity}_{f,m} = \gamma_f + \delta_m + \sum_{k=k_0}^{K} \lambda_k \widehat{\text{Takeup}_{f,m} \cdot \text{AI}^k_{f,m}} + \sum_{k \in \mathcal{C}} \beta_k \text{AI}^k_{f,m} + \theta_m X_{f,m} + \varepsilon_{f,m},$$

$$\text{Takeup}_{f,m} \cdot \text{AI}^k_{f,m} = \gamma^k_f + \delta^k_m + \sum_{j \neq -1} \pi^k_j \text{AI}^j_{f,m} + \theta^k_m X_{f,m} + \nu^k_{f,m}, \qquad k = k_0, \ldots, K.$$

Here, $\text{Takeup}_{f,m}$ is the share of the firm's active workers who have personally taken up the tool by month m. As in the main specification, $\text{AI}^k_{f,m} = \mathbb{1}\{m - \text{AI}_f = k\}$ indicates that month m is k months after the firm's adoption month AI_f. The first equation is the second stage. The second gives the first stages, one for each month k since adoption. Each first stage includes every event-time indicator. The indicators for months k_0 through K are the excluded instruments. The remaining indicators also enter the second stage as controls; as in the main specification, the indicators beyond the event window are binned. Both stages include the same fixed effects and controls as the main specification,

and we cluster standard errors by firm. Because the outcome is measured per worker, λ_k is the effect *per worker* who has taken up the tool, k months after adoption.[4]

Appendix Figure A8 displays the results. The first stage is strong for both tools ($F = 46.91$ for assistants and 49.46 for agents). For AI assistants (Panel A), the IV estimate at month 12 of adoption is 4,812.1 lines per adopting worker per month, or 139.7% of the baseline mean of 3,445.3. For AI agents (Panel B), lines of code added per adopting worker rise by 8,271.0 per month at month 12 of agent adoption, or 172.1% of the baseline mean of 4,806.1, compared with 44.0% at the firm level. Given that measured take-up is a lower bound, the true effect per user at month 12 lies between these two figures. Changes in the per-adopter effect over time could reflect learning, improvements in model quality, or changes in who adopts. We do not find a clear time pattern: the estimates are roughly flat. Consistent with this, the overidentification test in the pooled specification does not reject a constant effect per adopting worker across months (J, $p = 0.751$). That is, the rise in the firm-level effect is explained by the rise in individual-level take-up. It is possible that the different time dynamics within the per-adopter effect cancel one another out; or it is possible that our estimates are too noisy to capture this form of dynamic.

4.5 Effects on final software output

Next, we examine the impacts of this productivity shock on firm-level software output. As described earlier, we use two measures of final software output: the number of Jira issues resolved, where resolution captures deployment of code associated with a software task; and the number of Jira epics resolved, which represent completion of larger features or projects, of which an issue is a subcomponent. As Jira can be used to track tasks that are not software-related, we restrict attention to coding issues and epics, such that we can use these outcomes as measures of software output.

Figure 4 displays event study estimates for both measures. Panels A and C display the effects on these two outcomes for AI assistants, showing no significant effect on either: −0.027 additional issues resolved per worker-month, or −1.5% of the baseline mean of 1.80, and 0.003 additional epics, or 3.4% of the baseline mean of 0.095. Panels B and D display the effects for AI agents, with slightly larger increases of 13.7% on Jira issues and 18.0% on Jira epics, both of which are marginally insignificant. Our estimates rule out increases larger than 30.7% on issues and 41.5% on epics. One concern is that after AI adoption, teams may track issues less carefully in Jira, given the increased pace of code

[4]One caveat to the IV estimate is that we do not observe individual take-up perfectly, as described in Section F.1. In particular, for AI agents, we identify most individual users from the agents' traces in GitHub commits and pull requests. Workers can use the tool without leaving such a trace, so our measure of take-up — the first stage — is a lower bound on true take-up. In turn, the IV estimate will be an upper bound on the person-level productivity effect. However, it should not necessarily be the case that the bias in the first stage varies over time, so we should still consider the time dynamics of the figure as accurate.

writing. While we believe this to be a potential concern with using the raw issue counts, the epics should be less susceptible to such issues, given that it is less likely that teams would fail to record more substantial software projects.

A concern with the raw issue count is that it may mask changes in the composition of tasks — for instance, each task may grow in size, such that the issue count understates the growth in output. We test for this using our two measures of issue size described in Section 2.3: the LLM-assigned length label and the predicted log cycle time. Appendix Figure A11 plots event studies for average issue size under both measures. We find no evidence of a compositional shift in either case, suggesting that the null output result is not an artifact of task redefinition.

Finally, although software output may not change on the margin of quantity, it still may change on the margin of quality. To assess this possibility, Appendix Figure A12 examines the effects of AI adoption on the number of bug reports created in Jira, normalized by the number of workers per firm. We do not observe a significant change in quality after either tool's adoption. If anything, the point estimates are slightly negative, indicating a decline in the rate of bugs reported per worker, i.e., a slight increase in quality.

4.6 Effects on employment

Finally, we turn to the effects on employment. If firms do not fully pass through a productivity gain to increased output, they may instead adjust by cutting employment. We assess the effects on employment across three margins: (1) on the number of active engineers, as tracked in the JF data, and on total firm employment, as tracked through a merge with LinkedIn profiles; (2) separately for junior and senior workers; and (3) on the margins of hires versus separations.

We begin with the number of engineers active in the JF data in each firm-month, displayed in Figure 5. Neither AI assistants nor AI agents have a significant effect on engineering employment. However, these estimates are somewhat noisy — we can only rule out effects outside of −4.7% to 10.7% for assistants, and −15.6% to 10.5% for agents.

Next, we assess effects on the firm's total employment, as measured by the total number of LinkedIn profiles attached to the firm, in Appendix Figure A16. This measure allows us to assess effects on both engineering and non-engineering teams. It also allows us to assess heterogeneity by seniority, without issues of differential coverage by seniority. Both panels display zero effects. These estimates are more precise — we can rule out effects outside of −4.5% to 5.1% for assistants, and −2.5% to 10.3% for agents.

It is possible that the aggregate effect masks heterogeneity by worker seniority. Some papers have found employment declines among junior workers and gains among senior workers (Brynjolfsson, Chandar, & Chen, 2025; Lichtinger & Hosseini Maasoum, 2025). We test for this using the LinkedIn

data on junior and senior workers in the firm. The results are displayed in Appendix Figure A17. We do not find significant effects of either AI assistants or AI agents on junior or senior workers.

Finally, it is also possible that these aggregate effects mask churn in employment. In particular, the papers that have found employment declines have generally found them on the margin of reduced hiring. We assess this possibility using the LinkedIn data, measuring hires as the number of workers starting their first position at the firm each month, and separations as the number of workers ending their last position at the firm each month. Appendix Figure A16 displays the results for overall hires and separations, and Appendix Figures A18 and A19 display them separately for junior and senior workers. We find no evidence of a decline in hiring after either tool's adoption. AI assistants show a modest increase in hiring, driven by senior workers, and an insignificant change in separations. AI agents show null effects on both hires and separations. In particular, we do not find the decline in junior hiring documented in prior work. The one exception to the null effects on separations is a marginally significant increase in senior separations after AI agent adoption.

These null effects on employment are consistent with some empirical work documenting small employment effects of generative AI (Humlum & Vestergaard, 2025), though some papers find evidence of much larger effects (Brynjolfsson, Chandar, & Chen, 2025; Lichtinger & Hosseini Maasoum, 2025). We interpret the results of our staggered AI adoption design to rule out an employment decline that would result from a direct automation effect. In other words, we rule out the possibility that firms have integrated AI tools sufficiently into their workflows that they no longer need workers.

However, we also emphasize several caveats when interpreting our results. First, we present a reasonably short-run estimate of the effect of generative AI on employment; it is plausible that employment responds slowly to technological changes. Section 6 assesses potential future trajectories in employment outcomes. Second, our event study design identifies the partial equilibrium effect of generative AI adoption. It is plausible that this null partial equilibrium effect masks a negative general equilibrium effect. To see this possibility, note that giving a single firm access to a labor-augmenting technology will have two effects in partial equilibrium: first, there will be a *displacement effect*, since the number of workers per unit of output will fall; and, second, there will be a *scale effect*, as the output of the firm may increase (Acemoglu & Restrepo, 2018). This scale effect may be primarily business-stealing from other firms. If so, we would expect reductions in those firms' employment.

5 Production bottlenecks

We observe significant effects of AI coding tools on intermediate measures of code output, but we do not observe significant effects on downstream outcomes, including software output or employment. How can we reconcile such findings? In this section, we provide evidence on production bottlenecks

as a mechanism: if other steps in the software production process are slow to adjust, or negatively affected, productivity gains in code writing may not pass through to firm-level output. Our data provides us with particularly granular insights into the review process, which we focus on as a bottleneck.

5.1 Descriptives about the code review process

Software production requires a series of sequential complementary tasks, as described in Section 2. While AI raises productivity in code writing, other tasks may serve as limiting factors. In this subsection, we test whether code review serves as such a bottleneck.

Once a block of code is ready for review, the engineer will submit a GitHub pull request. One or several other team members will read through and perhaps write tests on the code, to assess whether the code meets the necessary usage requirements and integrates correctly into the rest of the codebase. If necessary, they will leave comments and submit a request for changes. The code writer and code reviewers will iterate through these changes until the code is ready to be combined into the codebase, at which point they will "merge" the pull request.

Workers' time allocation. To empirically test the relevance of production bottlenecks, we begin by documenting the relative breakdown of an engineer's time allocation into these distinct tasks in Figure 6 Panel A. Column 1 shows the results, imputing time allocation from the Jellyfish data. We convert the panel of work signals (GitHub activity, Jira activity, and Google Calendar activity) into relative time allocation on coding, code review, and management. With this approach, we observe that engineers spend only 40.8% of their time writing code. Column 2 shows the results from a survey of 100 Prolific engineers on their average weekly time breakdown. On average, engineers spend only 30.8% of their time writing code.

Substitutability of workers. Next, we provide evidence regarding the substitutability of workers between coding and review, in support of our modeling assumption of a review ceiling. Although firms do not hire completely separate code writers and reviewers, workers are not perfectly substitutable across these tasks. First, Figure 6 Panel B provides evidence on reviewers' familiarity with code. Reviewers mostly review code they already know: 82.8% of reviews are in a code base where the reviewer had previously committed code or opened a pull request. If each review had instead gone to a random pull request opened at the same firm in the same month, only 17.2% would have been. Reviewing draws on knowledge of a specific code base, so a worker cannot be moved onto reviewing any pull request at no cost. Second, Panel C provides evidence on the seniority of reviewers, separately by seniority of the code author. Senior engineers do 75.7% of the reviewing on pull requests written

by senior engineers, but only 32.8% on pull requests written by junior engineers. Put the other way, junior engineers do only 24.3% of the reviewing of senior engineers' code.

5.2 Effects on review

Figure 7 Panel A displays the impact of AI adoption on the length of the review process, measured as the number of days between when a pull request is submitted and when it is merged. AI assistants have no significant effect on review length (Panel A of Appendix Figure A14). We attribute this lack of effect to the fact that the impacts on coding activity, from the previous section, are only moderately large. By contrast, AI agents significantly lengthen the review process, by 1.99 days, or 56.1% of the baseline mean of 3.56 days.

AI may impact the review process through two channels. First, if code writing and code review are complementary steps in the production process, a speedup in code writing will lead to a buildup in code review if labor in code review is held fixed. Second, review productivity may have declined, because AI tools may decrease the quality of code entering review.

Panel C of Figure 7 provides evidence of changes on both of these margins. AI assistants again have no significant effect on either (Panel C of Appendix Figure A14). By contrast, AI agents significantly lengthen both the amount of time that code is in a backlog, by 1.67 days, or 59.9% of the baseline mean of 2.78 days, and the amount of time in active review, by 2.33 days, or 58.5% of the baseline mean of 3.98 days.[5]

Panel D of Figure 7 provides evidence on changes in the review process itself. The left half of the panel displays two measures of how often reviewers request changes on a pull request: the share of pull requests that a reviewer blocks from merging, with a formal request for changes, and the share that receive at least one comment. With AI assistants, the effect on the share of PRs that are blocked is small and insignificant (Panel D of Appendix Figure A14). With AI agents, the share increases by 1.07 percentage points, or 38.5% of the baseline mean of 2.77%, though this estimate is marginally insignificant. Furthermore, with AI agents, the share of pull requests that receive at least one comment increases by 4.61 percentage points, or 13.1% of the baseline mean of 35.17%, though this estimate is also marginally insignificant. The right half displays the number of review comments per pull request — i.e., the intensive margin of number of changes requested. AI assistants have no significant effect on this measure. AI agents increase it by 0.629 comments per PR, or 38.6% of the baseline mean of 1.63.

[5]As a note, these baseline means do not sum to the overall time to merge baseline mean, because they are computed over the subset of pull requests that have at least one comment, such that we can separate the length of the backlog time from the length of the review time. However, we choose to report the time to merge over all pull requests as our main effect, given that it captures a broader set of pull requests, and pull requests that do not receive a comment may still be reviewed.

Appendix Figure A20 displays effects on calendar activity, which we observe for the subset of firms that connect Google Calendar to JF. We find no noticeable effect on the overall number of meetings per worker: our estimates rule out changes outside −4.3% to 4.8% of the baseline mean of 27.89 meetings per worker per month for AI assistants, and outside −4.0% to 2.7% for AI agents. Management meetings — including one-on-ones and team meetings — which make up most of workers' calendars, are similarly unchanged. However, meetings dedicated to code review and non-work calendar events — blocks for out-of-office time, such as lunch or doctor's appointments — increase slightly, by 11.5% and 3.5% for AI assistants and by 6.0% and 4.3% for AI agents. Only the increase in non-work events for AI agents is statistically significant; the estimates for code review are imprecise, as meetings that we can detect as dedicated to code review are rare.

These patterns are consistent with engineers' accounts of how AI has changed their work. First, engineers describe AI-generated code as requiring more correction in review, particularly on more complex tasks. For instance, one senior software engineer describes:

> *For deeper engineering tasks, it doesn't really save as much time — it gets stuff wrong a lot, and you end up spending more time in the review process.*

Second, engineers describe their own work shifting from writing code towards reviewing it. One junior software engineer describes:

> *The model gets you maybe 95% of the way there, and then the remaining 5% is figuring out what it got wrong and fixing it. My work became less about writing code directly and more about reviewing and correcting what Claude produced.*

Finally, it might be the case that the number of comments or changes requested increases mechanically if PRs are just getting larger in size. We assess this possibility in Appendix Figure A15. We do not observe a significant change in the size of each pull request, measured by the lines of code added in the pull request, either for AI assistants or for AI agents.

5.3 Usage of AI in code review

One concern may be that this code review bottleneck will be alleviated with deployment of AI tools to perform code reviews. Thus, we examine the extent to which firms deploy AI directly in the code review process. AI coding tools are technically capable of performing code review tasks in addition to writing code. However, firms may choose to keep humans in the loop at the review stage even if AI assistance is available, given the high cost of deploying faulty code.

Figure 8 documents the extent to which AI tools are used directly in code review. Panel A displays firm-level take-up of AI code review tools over time. Take-up of AI in the code review process is high: as of July 2026, 79.1% of firms had used AI tools in the code review process. Panel B examines usage

of AI on the intensive margin and the extent to which humans remain involved. It shows that only 23.3% of all review comments are generated by AI, and only 10.8% of pull requests receive at least one AI-generated comment.

Together, these figures indicate that although the majority of firms have begun to use AI tools in the code review process, humans are still heavily involved with review. This evidence is consistent with anecdotal accounts of changes in the review process from integrating AI. For instance, one senior engineer remarked:

> *Even with AI tools, you absolutely still need humans involved in the review process. I don't see a time where you will ever not need human oversight, even as tools get better. The AI tools lack the comprehension of the overview of the entire feature, and what it is supposed to do.*

5.4 Heterogeneity by worker and firm characteristics

Finally, we examine heterogeneity in code output, the bottleneck, and final software output by worker and firm characteristics. We consider three worker characteristics — (1) baseline coding activity, (2) seniority, and (3) software area — and five firm characteristics — (1) take-up intensity, (2) firm size, (3) industry, (4) software product type, and (5) codebase complexity.

We estimate the following specification, with outcomes collapsed to the level of firm f, group g, and month m:

$$\text{Activity}_{f,g,m} = \gamma_{f,g} + \delta_m + \sum_{k \neq -1} \beta_k^g \, \text{AI}_{f,m}^k + \theta_m X_{f,m} + \varepsilon_{f,g,m}.$$

Here, β_k^g is the effect of AI adoption for group g. As in our main specification, we include month fixed effects δ_m and time-varying controls $X_{f,m}$ — baseline firm size and the time since the firm's data coverage began. For worker characteristics, each worker is assigned to one group — e.g., by seniority or baseline coding activity — outcomes are aggregated to the firm-by-group-by-month level, and $\gamma_{f,g}$ is a firm-by-group fixed effect. For firm characteristics, each firm is assigned to one group — e.g., by firm size or codebase complexity — so the panel is firm-by-month and $\gamma_{f,g}$ reduces to the firm fixed effect γ_f. Otherwise, the specification follows our main specification: we bin event times beyond the window, restrict to firms that adopt during the sample period and have both GitHub and Jira data, and cluster standard errors at the firm level. In the heterogeneity figures, we report $\hat{\beta}_{\text{last}}^g$ — the effect in the final month of the event window — as a percent of each group's pre-adoption mean, and test whether it is equal across groups.

5.4.1 Worker characteristics

We begin with the three worker characteristics, shown in Appendix Figures A21 and A22. The effects of AI are largest for workers with the least coding activity at baseline. Consistent with this, the effects are slightly larger for senior workers, who code less at baseline, than for junior workers, though the difference is small and not significant. The effects vary little across software areas. Turning to the bottleneck, we find little heterogeneity in review outcomes by who writes the code.[6]

5.4.2 Firm characteristics

We next turn to firm characteristics. Appendix Figures A23 and A24 show the effects by take-up intensity — the share of a firm's engineers who use the tool. For AI assistants, the effect on pull requests per worker is similar in the bottom two terciles (23.8% and 23.1%) and larger in the top (41.8%). For AI agents, the effects on code output rise steeply with take-up: pull requests per worker increase by 21.4% in the bottom tercile, 41.9% in the middle, and 67.0% in the top. Appendix Figures A25 and A26 show the effects by firm size, industry, and whether software is the firm's product or an input to it. None of these dimensions shows differences that are consistent across the two arms.

Finally, Appendix Figures A27 and A28 show the effects by codebase complexity, measured by the number of code repositories and the number of branches. This provides a test of the bottleneck hypothesis: in firms with more complex codebases, it is plausibly harder to reallocate workers to review code they did not write. For AI agents, the patterns are consistent with this hypothesis. In the top tercile of repositories, time to merge rises by 69.7% — against 25.4% in the bottom tercile — while pull requests per worker rise by 30.0%, against 64.3%. Branches give a similar gradient (Appendix Figure A28, Panel B). The effects on Jira issues resolved are similar across terciles.

6 Model estimation and future impacts

In this section, we bring our empirical estimates from Sections 4 and 5 to our model of software production from Section 3, in two steps. We begin by estimating the model, to recover effects that we cannot observe directly. Although we observe the effects of AI adoption on intermediate outputs (e.g., code written; reviews completed) and final outputs (software projects completed), we cannot directly measure a "productivity" effect — i.e., the rate at which labor inputs are converted into these outputs. Similarly, although we observe employment effects on the extensive margin, we cannot observe how firms reallocate labor internally between coding and review. The estimation also allows us to assess how much of the gap between the changes in intermediate and final software output is driven by the

[6]Not every worker can be classified on each characteristic, so the group estimates need not average to our main estimate.

review bottleneck, versus by a change in the number of code changes per unit of output (i.e., the change in n).

We then use the estimated model to simulate the effects of AI over a longer time horizon, in which the firm may adjust labor more flexibly, or in which AI tools may advance in their capabilities.

6.1 Set-up

As before, denote the firm's baseline optimal value of a variable X as X^*, the post-AI adoption observed value as X', and the post-AI adoption optimal value as $X^{*\prime}$. Now, we also denote the log change in an outcome as $\mathrm{d}\ln X \equiv \ln X' - \ln X^*$.

Moments and constraints. Our reduced form event studies provide us with estimates for $\hat{m} = (\mathrm{d}\ln C^P, \mathrm{d}\ln R^P, \mathrm{d}\ln Y, \dot{B}, \dot{W}, \mathrm{d}\pi)$. We also measure the firm's baseline optimal values $\psi = (L_C^*, L_R^*, C^{P*}, R^{P*}, n^*)$. We would like to obtain values for $\theta = (\mathrm{d}\ln A_C, \mathrm{d}\ln A_R, \mathrm{d}\ln L_C, \mathrm{d}\ln L_R, \mathrm{d}\ln n)$. The firm's problem provides us with the following six moments, which allow us to derive these unobserved parameters from the observed data.

$$
\begin{aligned}
&\text{(M1) Code produced} && \mathrm{d}\ln C^P = \mathrm{d}\ln A_C + \mathrm{d}\ln L_C,\\
&\text{(M2) Reviews produced} && \mathrm{d}\ln R^P = \mathrm{d}\ln A_R + \mathrm{d}\ln L_R,\\
&\text{(M3) Review productivity} && \mathrm{d}\ln A_R = \ln[1-\pi'] - \ln[1-\pi],\\
&\text{(M4) Software output} && \mathrm{d}\ln Y = \mathrm{d}\ln R^P - \mathrm{d}\ln n,\\
&\text{(M5) Backlog slope} && \dot{B} = C^{P*}\left[e^{\mathrm{d}\ln C^P} - e^{\mathrm{d}\ln R^P}\right],\\
&\text{(M6) Wait slope} && \dot{W} = e^{\mathrm{d}\ln C^P - \mathrm{d}\ln R^P} - 1.
\end{aligned}
$$

And it provides the following two constraints:

$$
\begin{aligned}
&\text{(C1) Labor adds up} && L_R^*\left[e^{\mathrm{d}\ln L_R} - 1\right] = -L_C^*\left[e^{\mathrm{d}\ln L_C} - 1\right],\\
&\text{(C2) Review ceiling} && L_R^{*\prime} \;\le\; \bar{L}_R = \phi/\rho.
\end{aligned}
$$

We provide a description of how we convert the model set-up from Section 3 into these moments in Appendix Section G.

Estimation. We estimate θ by classical minimum distance (Chamberlain, 1984; Newey & McFadden, 1994): we choose the parameters that bring the model's moments closest to the moments we estimate

in the data,

$$\hat{\theta} = \arg\min_{\theta} [m(\theta) - \hat{m}]' W [m(\theta) - \hat{m}].$$

Here, $\hat{\theta} = (\widehat{\mathrm{d}\ln A_C}, \widehat{\mathrm{d}\ln A_R}, \widehat{\mathrm{d}\ln L_C}, \widehat{\mathrm{d}\ln L_R}, \widehat{\mathrm{d}\ln n})$ is our vector of estimated parameters, $m(\theta)$ is the vector of the model's moments for a given θ, and $\hat{m}$ is the vector of our event-study estimates. W is a weight matrix, with the inverse squared standard errors of the corresponding event-study estimates along its diagonal. We provide a description of the empirical estimates we use as well as the implementation of the estimation in Appendix Section G.

6.2 Results and test of model fit

The results of our estimation are reported in Table 5. First, our estimation allows us to assess the reduced form estimates' implied changes in coding and review productivity. Coding productivity increases significantly, at 73.7%. This is larger than what is implied by the pull requests created, because labor is being reallocated away from coding at the same time: the same flow of created requests is produced by fewer coder-hours. Additionally, review productivity decreases slightly, at −7.0%, which comes from the increase in the share of requests that draw a comment, though the effect is marginally insignificant.

The estimates also provide implied values of the intensive margin of labor reallocation. They indicate a reallocation away from coding labor of −22.7% (−14.9 percentage points from a baseline share of 0.657) and towards review labor of 43.4% (14.9 percentage points from a baseline share of 0.343).

Finally, the change in n captures the residual difference between the code output and final output numbers, that is, the number of code changes per unit of final output. There are several ways in which we can interpret this number. First, this could capture an AI-induced behavioral change: if engineers change the frequency with which they commit or open pull requests, then n would increase. Second, it could capture other forms of bottlenecks that emerge from AI adoption: for instance, bottlenecks from the management or testing process. We observe an increase of 17.3%.

Taking these estimates together, we observe a pass-through rate of the coding productivity shock to final output of 0.233 in log points: an increase in coding productivity of 73.7% leads to an increase in final output of 13.7%. More specifically, we can break down the pass-through of the coding gain to final output as follows. Of the difference between these outcomes, 62.4% can be explained by the production bottleneck — labor withdrawn from coding, together with the code that accumulates in the backlog rather than being merged — and 37.6% by the residual fragmentation term.

The table also reports the test of model fit. We have six moments and five parameters, one

of which is derived from another through the labor adding-up constraint, so there are four free parameters and two degrees of freedom. Specifically, the place in which the model is overidentified is the queueing structure: the backlog slope and the wait slope are the two moments that no parameter is fitted to, and to first order they jointly require $\dot{B}/C^{P*} = \dot{W}$. We obtain a J statistic of 2.91, with a p-value of 0.233, indicating that we cannot reject the model.

6.3 Future impacts

Our empirical estimates provide us with evidence on the short-run impacts of AI on productivity, output, and employment, with current versions of AI tools and short-run firm adjustment. However, we are also interested in how AI will impact these outcomes in the future. With the estimated model, we explore counterfactual changes in outcomes: first, changes from additional firm adjustment; second, changes from advancements in AI's technical capabilities. In doing so, we follow recent work that uses models of production to assess the economic implications of advances in AI (e.g., C. Jones, 2026; C. Jones & Tonetti, 2026; B. Jones, 2025; **?**).

6.3.1 Firm adjustment

We explore two ways in which the firm may continue to adjust, even without additional advancements in AI capabilities. First, we explore what additional adjustment is possible with only internal labor reallocation, and without any additional hiring or labor displacement. Second, we explore how the firm may choose to respond in the longer run with more flexible labor adjustment and output choice — i.e., if we relax the assumption that overall labor is held fixed.

Medium-run adjustment, with fixed overall labor. To assess how much the firm can adjust internally, we calculate the labor allocation that would balance the coding and review capacities, obtaining a split of $L_C^{*\prime} = 0.506$ and $L_R^{*\prime} = 0.494$. The allocation our estimates imply is 0.508 and 0.492, so the firm has only a small amount of reallocation left to do. We then calculate the output the firm would produce if it moved to this balanced allocation.[7] It would produce 2.91 resolved issues per worker-month, or 14.1% above the pre-period, against the 13.7% we measure. Thus, further internal reallocation adds little to output. Instead, much of the gap between the 73.7% increase in coding productivity and the increase in output comes from the change in the residual term n, which rises by 17.3%. If n had stayed at its pre-adoption level, the balanced allocation would produce 33.8% more output than in the pre-period. One interpretation is that part of the increase in A_C does not represent "useful" work that passes through to output; another is that there are other bottlenecks —

[7]Here, we hold n fixed at the post-agent adoption value that we measure, so that we can assess the impacts just from further labor reorganization.

e.g., in management or testing — that reallocating labor towards review cannot resolve; if these other bottlenecks did not exist, we would observe this 33.8% increase in output.

Long-run adjustment, with flexible labor. In the longer run, the firm may choose to hire or displace workers, rather than just reorganizing labor internally. Thus, in line with Section 3 Proposition 5, we relax the assumption that overall labor is fixed. Because the firm faces constant returns to scale, we must impose some additional structure in order to pin down the way that the firm will set its level of overall output and employment. Thus, we introduce the product demand function that the firm faces. The price elasticity of this function ε will govern the extent to which the firm decides to pass through the productivity shock to output versus employment.

The proposition provides us with comparative statics, linking the changes in output and employment to the elasticity of product demand. We can combine these comparative statics with our empirical estimates to characterize both the slope of each relationship and the threshold ε above which an outcome rises and below which it falls. The details of this exercise are described in Appendix Section G.3.1. In this exercise, we hold n at its pre-adoption level.

Figure 9 plots the four responses against ε. Output and review labor are always rising. However, overall labor and coding labor may fall or rise depending on ε. The threshold for overall labor to increase is $\varepsilon = 3.68$ and for coding labor is $\varepsilon = 6.83$.

To provide a sense of which scenario is most likely, we bring in estimates of the elasticity of software product demand from the existing literature. The papers from which these estimates come are detailed in Table 6; they estimate demand for a range of software products — mobile applications, retail software, and console video games — using a range of approaches, including BLP demand estimation. While each of these software products differs from the products in our setting, and the elasticity also varies along the demand curve, we think of the range as still informative. Furthermore, we observe that for most of the range of existing estimates, we would expect to see a decline in overall labor. At the mean of the estimates, we would expect to observe an increase in output of 16.1% and a decrease in overall employment of 13.2%. Specifically, this decline comes from a fall in coding employment of 33.2%, while review labor still increases by 24.9% — demonstrating that the labor reallocation goes towards the bottleneck. In other words, the model implies that the firm ships more with fewer engineers, and that engineering work shifts towards review rather than code writing.

6.3.2 Advances in AI

The previous section explored how the firm may continue to adjust, holding the technology fixed. In this section, we explore how advances in AI's technical capabilities may shape the trajectories of coding productivity, output, and employment.

Trajectories of technical capabilities and coding productivity. We begin by projecting a trajectory for coding productivity A_C. Our estimates give us two points on this trajectory. AI assistants, adopted by the median firm in April 2024, raise A_C by a factor of 1.35 relative to the pre-AI period; AI agents, adopted by the median firm in May 2025, raise it by a further factor of 1.74, for a cumulative factor of 2.34.[8] To extend this trajectory beyond our two estimates, we need to take a stance on its shape — e.g., linear versus exponential. We draw on recent work in computer science that tracks the technical capabilities of frontier AI models over time. In particular, Kwa et al. (2025) measure the length of software tasks — in terms of the time a human expert takes to complete them — that frontier models can complete with 50% reliability. They find that this time horizon has grown exponentially (i.e., linear in logs), doubling roughly every 129 days for models released since 2023, and use this trend to forecast future capabilities. We therefore fit the same type of curve to A_C: we assume that A_C similarly grows exponentially — i.e., that $\ln A_C$ rises linearly over time, at a different rate — and fit this curve through our two estimates of the productivity gains from adoption of AI assistants and AI agents. Appendix Section G.3.2 describes this construction in detail.

Figure 10 Panel A plots the results of this exercise. The grey line shows the time horizon of frontier models relative to its level in February 2023, with METR's measurements marked; the red line shows the fitted path for A_C, with both extrapolated through 2031. Two things stand out. First, the slope of technical capabilities on lab benchmarks is far steeper than the slope of productivity gains in practice. Between the median adoption dates of AI assistants and AI agents, the time horizon rises 8.43-fold, while coding productivity rises 1.74-fold. Second, if technical capabilities continue to improve along this trajectory, A_C would rise to 8.93 times its pre-AI level by 2028 and 41 times by 2031.

These are very large numbers. However, they are not unprecedented relative to earlier technological changes. For example, computing performance improved at an average rate of roughly 45% per year in the decades after 1945 (Nordhaus, 2007). In cotton spinning, the labor needed to spin 100 lb of fine yarn fell from over 50,000 hours by hand to about 2,000 hours on Crompton's mule in 1780, and to about 300 hours by 1795 (Catling, 1970; Chapman, 1972).

Trajectories of output and employment. Panels B and C of Figure 10 plot the model-implied trajectories of output and employment along this path for A_C, under two scenarios for review productivity: A_R held fixed at its estimated level (Panel B), and A_R growing as fast as A_C (Panel C). In both scenarios, the firm divides its labor between coding and review optimally at each point in time. There are several modeling decisions we must make in order to generate this forecast. Most importantly, we hold overall labor fixed until the end of 2026, consistent with the null employment

[8] We stack the two estimates because firms that adopt AI agents have typically already adopted AI assistants, so the agent estimate is measured relative to a pre-period in which assistants were already in use.

effects we estimate over the window of our data, and the propositions for the short- and medium-run impacts of AI adoption. From 2027, we allow overall labor to adjust toward its long-run level at the mean demand elasticity from Figure 9. The trajectories through 2026 therefore show what would occur with internal reallocation alone; the trajectories from 2027 show what would occur if the firm decides to hire or displace workers. Appendix Section G.3.3 describes the construction of these trajectories in detail, including the modeling assumptions involved.

There are three things to note. First, the gains in output are far smaller than the gains in coding productivity. By 2031, the projected 41-fold increase in A_C translates into a 57.8% increase in output if A_R is fixed, and a 341.4% increase if A_R keeps pace. In the short run, this lack of pass-through is due to labor inputs being difficult to scale. In the long run, it is due to the demand for software products constraining firms' output growth, so that firms pass through part of the productivity gain to an employment cut instead. Second, consistent with review acting as a bottleneck, the trajectories depend heavily on whether review productivity improves. With A_R fixed, output levels off at about 57.8% above its pre-AI level by late 2027. By late 2028, the share of labor devoted to review reaches its ceiling, and further gains in coding productivity add only to the backlog of unreviewed code. With A_R growing as fast as A_C, by contrast, output keeps rising — by 140.9% by the end of 2026 alone. Third, larger gains in output come with larger labor displacement. With A_R fixed, employment falls by 10.3% by 2031, and stops falling once the review ceiling binds: beyond that point, further gains in coding productivity no longer reduce the labor needed per unit of output. With A_R growing as fast as A_C, each unit of output requires less labor in both coding and review, and employment falls by 76.2% by 2031.

We view these scenarios as bounds on the long-run impacts of AI. The scenario with A_R fixed is one in which AI can do only approximately half of software work — writing code — while the rest of the work still requires human labor. The scenario with A_R growing as fast as A_C is one in which AI can do essentially all of software engineering work, since in our model, software production involves only these two inputs. The magnitudes in this second scenario are therefore very large, and should be read as an upper bound. In practice, software production also involves tasks that our model abstracts from — e.g., deciding what to build, testing, and coordinating with customers — which would act as further bottlenecks.

7 Conclusion

Generative AI has garnered significant attention, yet its impacts on real-world production settings are still poorly understood. This paper studies firm-level adoption of two generations of AI coding tools — AI assistants and AI agents — using 445 million work events from an engineering analytics

platform, which allow us to observe the full software production process, from code writing to code review to final software output.

We find that AI tools substantially increase intermediate code output, but these gains do not translate into proportional changes in final software output or employment. Following agent adoption, pull requests opened increase by 35.7%, while Jira issues resolved increase by only 13.7%, and we can rule out increases larger than 30.7%. We cannot attribute significant changes in engineering employment to either generation of tools.

Code review explains much of this gap. After agent adoption, the time from a pull request's submission to its approval increases by 56.1%. Code waits longer for review — consistent with a capacity constraint — and spends longer in active review and draws more comments — consistent with a decline in code quality. Estimating our model with these estimates as moments, we find that agents raise coding productivity by 73.7%, and that firms shift labor from coding towards review.

The estimated model also suggests that these short-run effects are not the end of the story. Once firms adjust employment, the model predicts that output rises by 16.1% and employment falls by 13.2%. With continued technological advancement, the long-run effects depend crucially on whether AI raises review productivity as quickly as coding productivity: if it does not, output rises by 57.8% by 2031; if it does, output rises by up to 341.4% and employment falls by up to 76.2%.

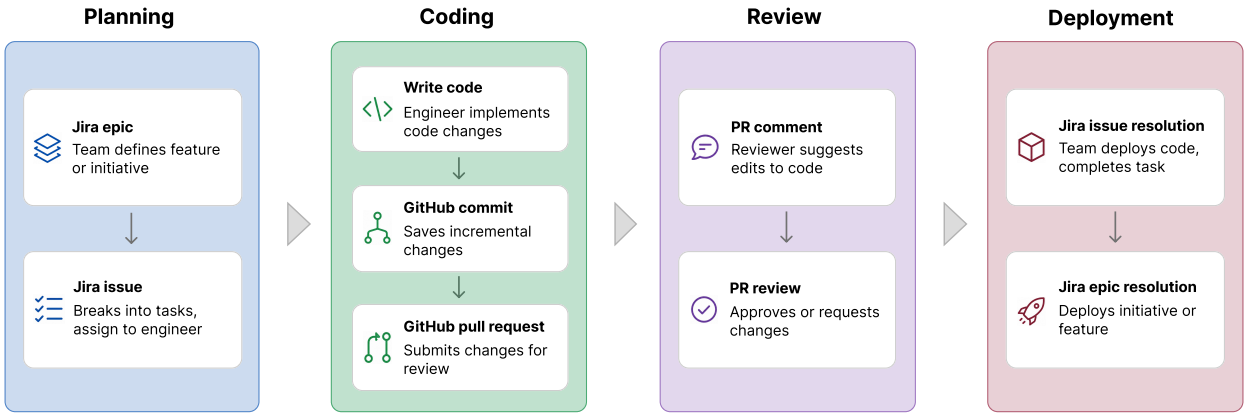
These findings have several implications for firms. First, AI's productivity impacts are uneven across tasks, even within an occupation like software engineering that is considered highly automatable. Thus, measuring the returns to AI requires evaluating the entire production process: intermediate metrics, such as lines of code or commits, overstate the gains from AI adoption. Second, realizing these gains requires complementary organizational investments. As one stage of production speeds up, the bottleneck shifts to another, and labor must be reallocated towards it.

Our results are subject to several qualifications. First, we estimate effects over the first 12 to 16 months after adoption, and employment may respond slowly to technological change. Second, our design identifies partial equilibrium effects; if output gains come partly from business-stealing, employment at other firms may fall. Third, our sample consists largely of medium-to-large, well-established firms, and effects may differ for start-ups or in other knowledge work settings.

More broadly, our findings suggest that the economics of AI adoption cannot be understood by studying automated tasks in isolation. As with previous waves of technological change, the impacts of AI depend not only on the technology but also on how quickly firms reorganize production around it. Understanding where bottlenecks arise — and how firms should reorganize to address them — will be central to understanding the long-run effects of AI on productivity, organizational design, and labor demand.

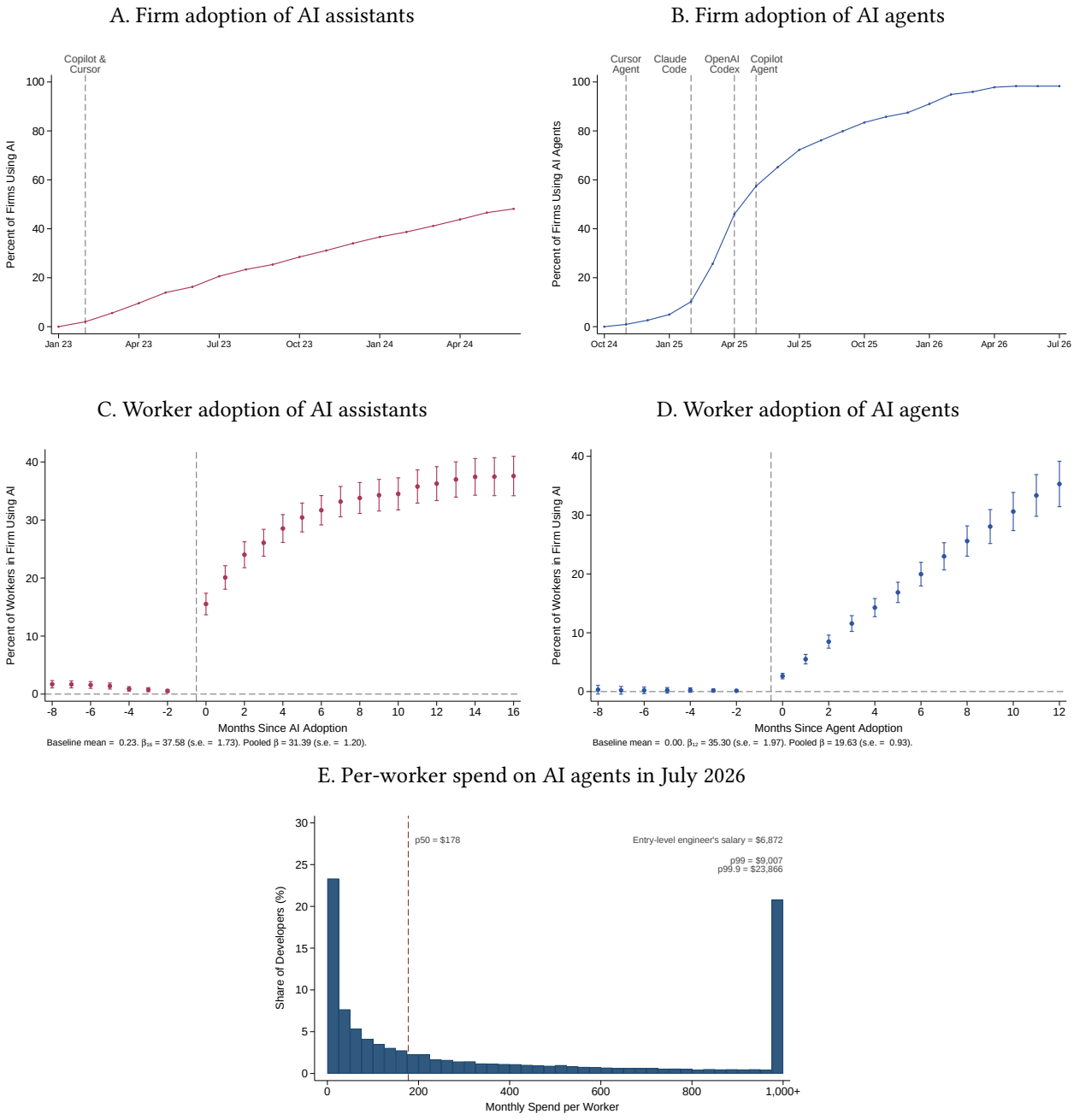
A Figures

FIGURE 1. Software production process



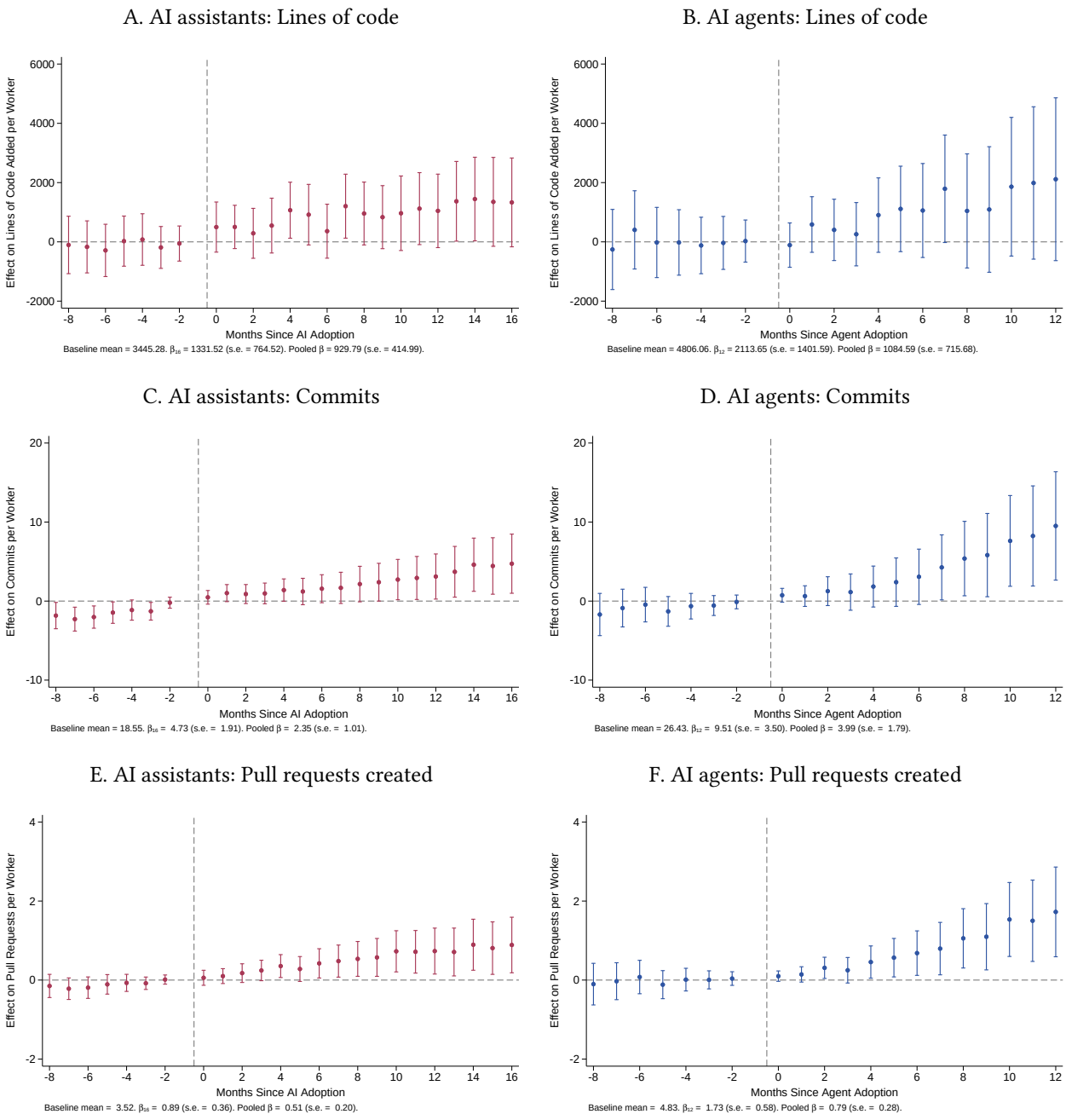
Notes: This figure provides an overview of the software production process, including a description of how our observed work signals fit into the production process.

FIGURE 2. AI adoption



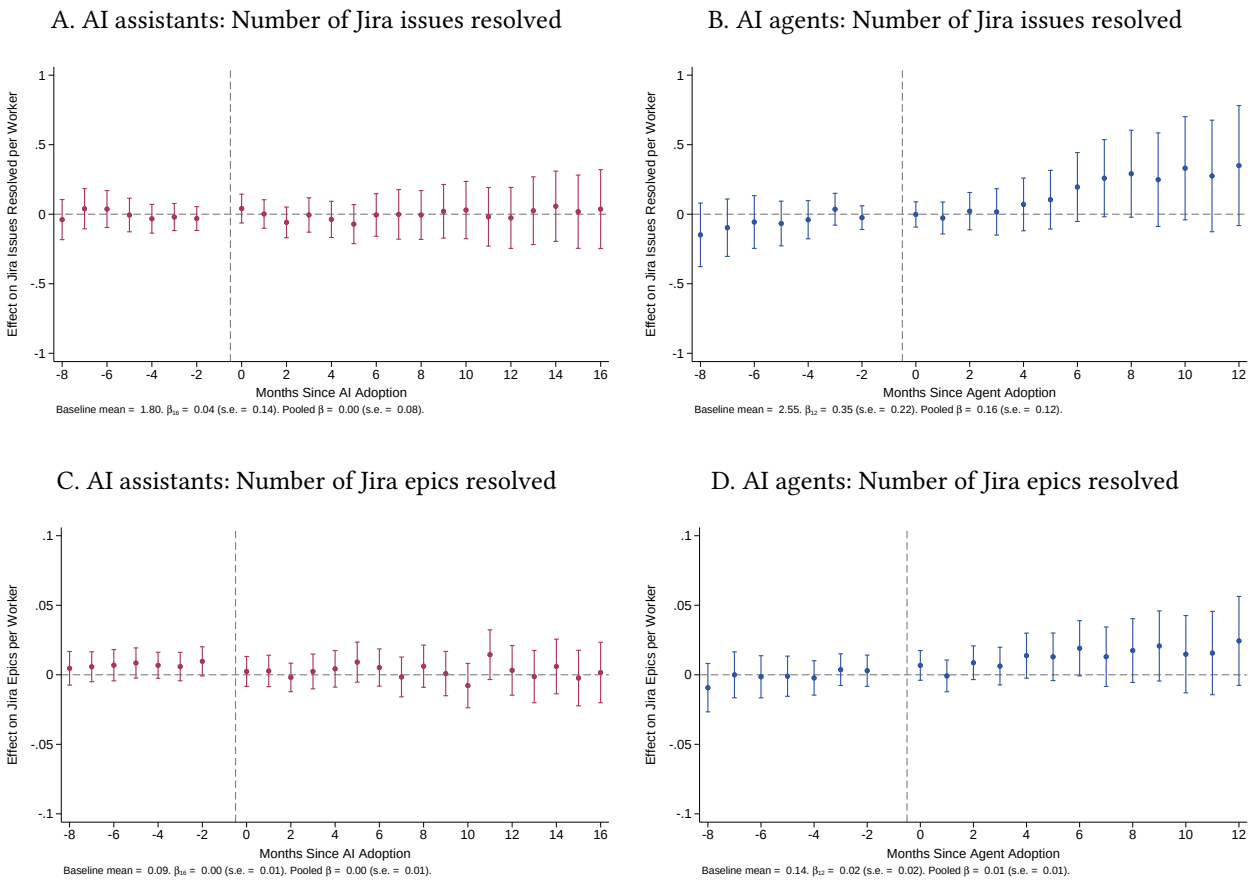
Notes: This figure displays information on firm-level and worker-level adoption of AI coding assistants and AI agents. Panel A displays the share of firms in our sample that have adopted an AI coding assistant (GitHub Copilot or Cursor) over time, beginning with the release of Copilot's business license and Cursor in February 2023. Firm adoption date is measured as the first month in which a firm purchased seats through the GitHub Copilot or Cursor licensing APIs. Panel B displays firm-level adoption of AI agents over time, beginning in October 2024. Vertical lines mark the release of agentic features in Cursor (November 2024), Claude Code (February 2025), and GitHub Copilot Agent (April 2025). Firm adoption date is measured as the earliest of: first agent activity detected through the Claude Code API, or first instance of any form of agent activity in the commits and pull requests, as described in Section 2.4. Panels C and D display worker-level adoption of AI assistants and AI agents, respectively, in the months elapsed since the firm's first adoption. The take-up rate is calculated among individuals who showed some level of GitHub activity in the baseline window. Panel E displays the per-worker spend on AI agents — measured through the Claude Code and Cursor APIs — in July 2026.

FIGURE 3. Effects of AI on code output



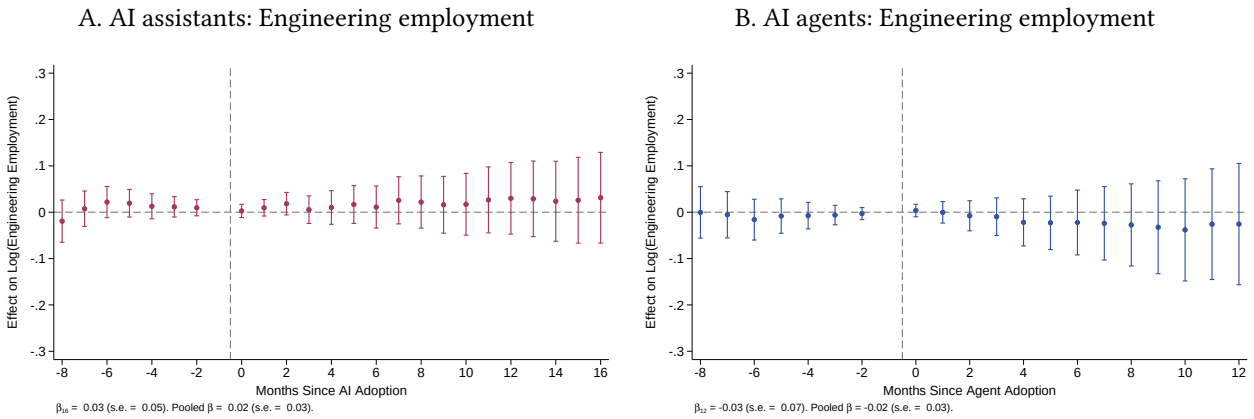
Notes: This figure displays event studies of the effects of AI adoption on coding productivity. In Panels A and B, the outcome variable is the number of lines of code added, normalized by the number of workers. We construct a firm-by-month panel, where the outcome variable in each row is the number of lines of code per worker-month — i.e., the total number of lines of code added by the firm that month, divided by the number of engineers active in JF in that firm-month. In Panels C and D, the outcome variable is the number of commits added per worker. In Panels E and F, the outcome variable is the number of pull requests created per worker. In Panels A, C, and E, the treatment is adoption of AI assistants. In Panels B, D, and F, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE 4. Effects of AI on software output



Notes: This figure displays event studies of the effects of AI adoption on software output. In Panels A and B, the outcome variable is the number of Jira issues resolved, normalized by the number of workers. We construct a firm-by-month panel, where the outcome variable in each row is the Jira issues resolved per worker-month — i.e., the total number of Jira issues resolved by the firm that month, divided by the number of engineers active in JF in that firm-month. In Panels C and D, the outcome variable is the number of Jira epics resolved per worker. In Panels A and C, the treatment is adoption of AI assistants. In Panels B and D, the treatment is firm-level adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

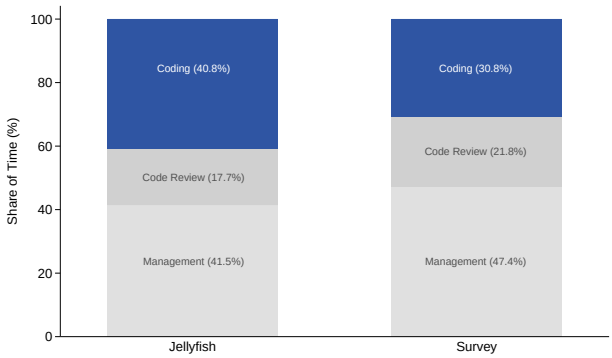
FIGURE 5. Effects of AI on employment



Notes: This figure displays event studies of the effects of AI adoption on log employment. In Panels A and B, the outcome variable is engineering employment, measured as the number of workers who are active in the JF data in each firm-month. In Panel A, the treatment is adoption of AI assistants. In Panel B, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

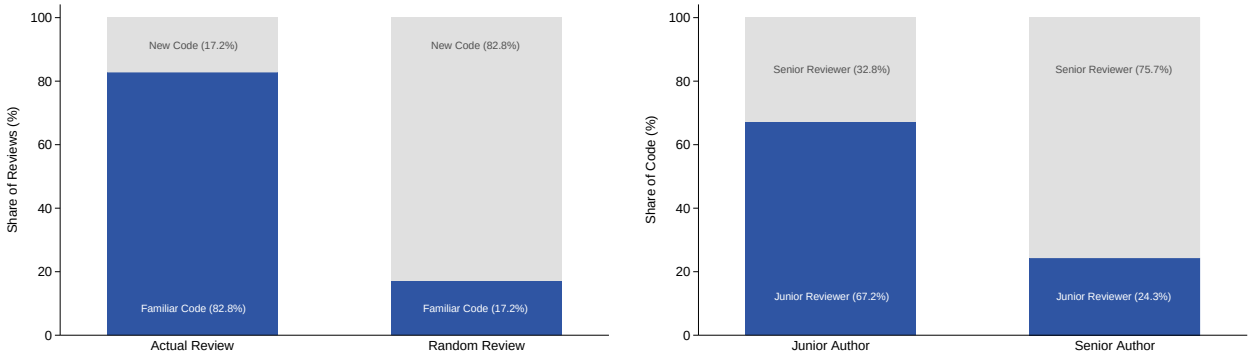
FIGURE 6. Code review process

A. Time allocation, pre-AI



B. Reviewer familiarity with code reviewed

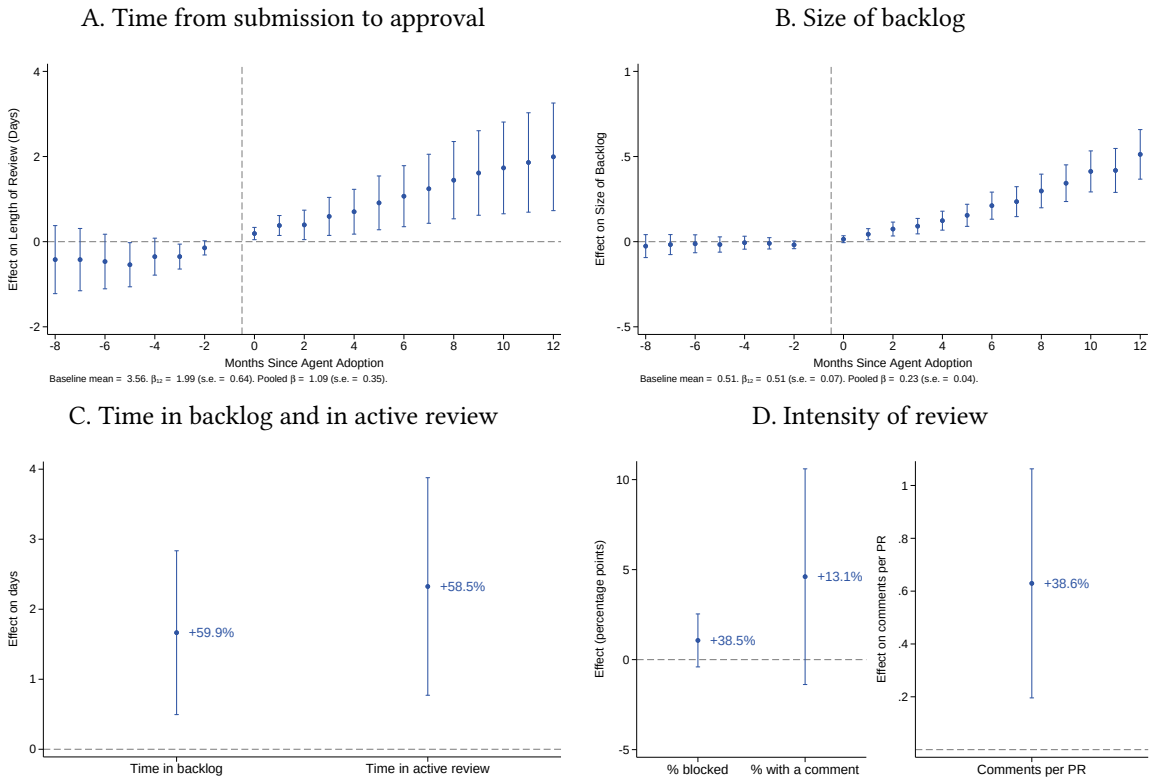
C. Reviewer seniority by code author seniority



Notes: This figure provides descriptives about the code review process.

- Panel A displays software engineers' average time allocation across three categories of tasks: coding, code review, and management. The first column displays results based on an imputation of time allocation from work signals in the Jellyfish data. We construct a panel of all timestamped work signals per worker and assign each signal to one of three categories: coding (commits, pull requests submitted, Jira issues resolved), code review (pull requests reviewed or merged), or management (Google Calendar meetings, Jira planning activity). We then impute the time spent on each signal as the elapsed time since the prior signal, truncated at two hours. We plot the average share of time in each category across workers. The second column displays results from a survey of 100 software engineers recruited through Prolific. We ask engineers to assess the percent of time they spend on each category in a typical week, grouping planning, logistics, and other responses into a single management category.
- Panel B displays the breakdown of reviews in code bases that the reviewer is familiar with, versus code bases that are new to the reviewer. The unit of observation is a review on a pull request. A code base is "familiar" if the reviewer had previously committed code or opened a pull request in it and "new" if not. The first bar shows the breakdown of familiarity of actual reviews. The second bar shows a benchmark for comparison: the share that would be familiar if each review had gone to a random pull request opened at the same firm in the same month, excluding the reviewer's own pull requests.
- Panel C displays the breakdown of seniority of reviewers, by the seniority of the code author. The unit of observation is a pull request. The first bar displays the seniority of reviewers on pull requests that are authored by junior workers, and the second for pull requests authored by senior workers.

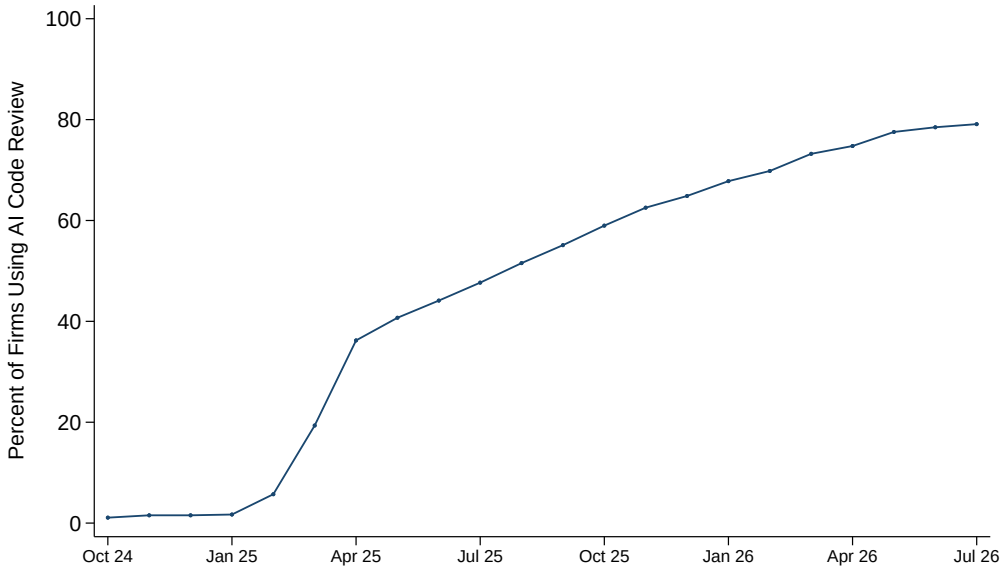
FIGURE 7. Effects of AI agents on code review



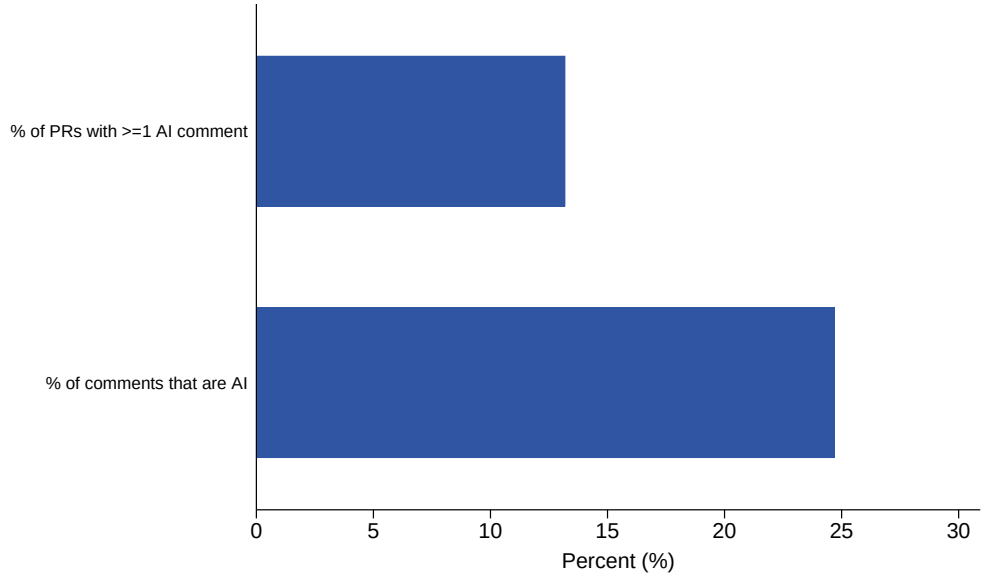
Notes: This figure displays the effects of AI agent adoption on the code review process. Panel A displays an event study of the length of review, the number of days between when a pull request is submitted and when it is merged. Panel B displays an event study of the size of the backlog, the number of PRs open at the firm when a new PR is added. Panels C and D display $\hat{\beta}_{12}$, with its 95% confidence interval; the label beside each point gives $\hat{\beta}_{12}$ as a percent of the outcome's pre-adoption mean. In Panel C, review length is split into time in backlog, the number of days from when a pull request is submitted to its first comment from a human other than its author, and time in active review, the number of days from that first comment to merge. In Panel D, the left half displays the share of pull requests for which at least one reviewer submitted a formal request for changes, and the share of pull requests that received at least one comment from someone other than the author, both in percentage points; the right half displays the number of review comments per pull request. In all panels, the treatment is adoption of AI agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE 8. Usage of AI in code review

A. AI code review tool take-up

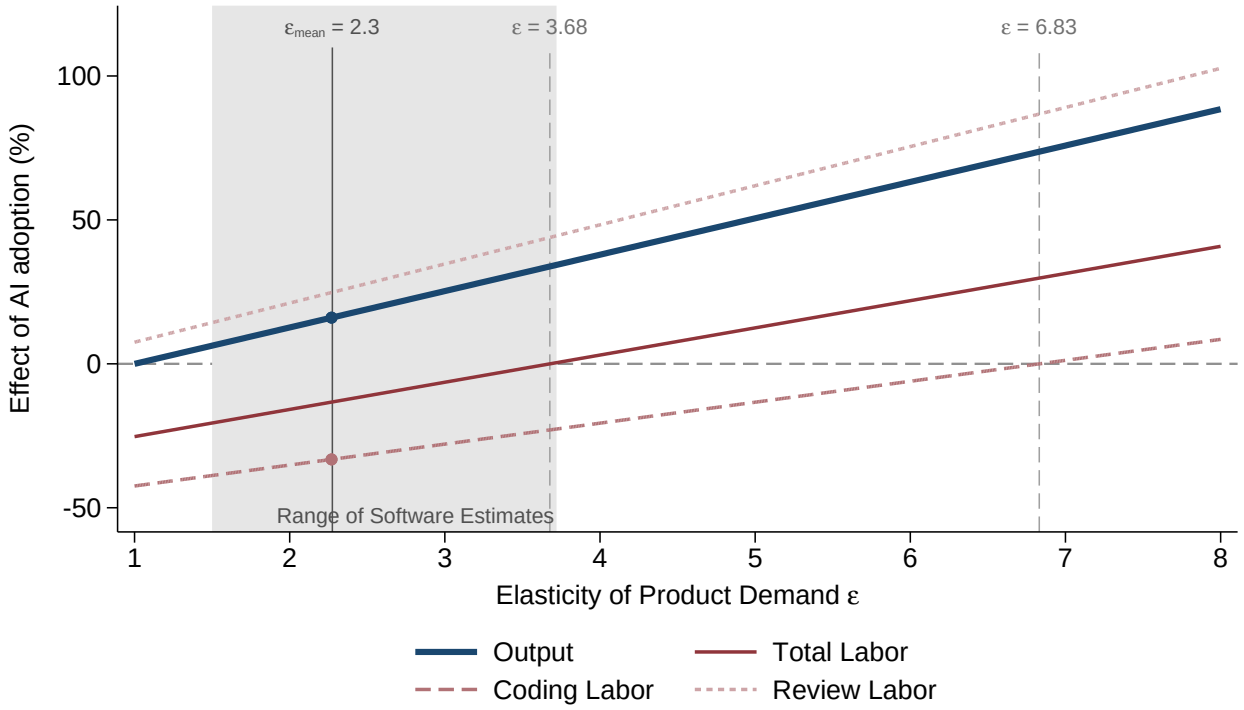


B. Rate of AI comments in reviews



Notes: This figure documents the extent to which AI tools are used directly in the code review process. Panel A displays the share of firms in our sample that have adopted an AI code review tool over time. Panel B displays the share of all review comments that are generated by AI, and the share of pull requests that receive at least one AI-generated comment.

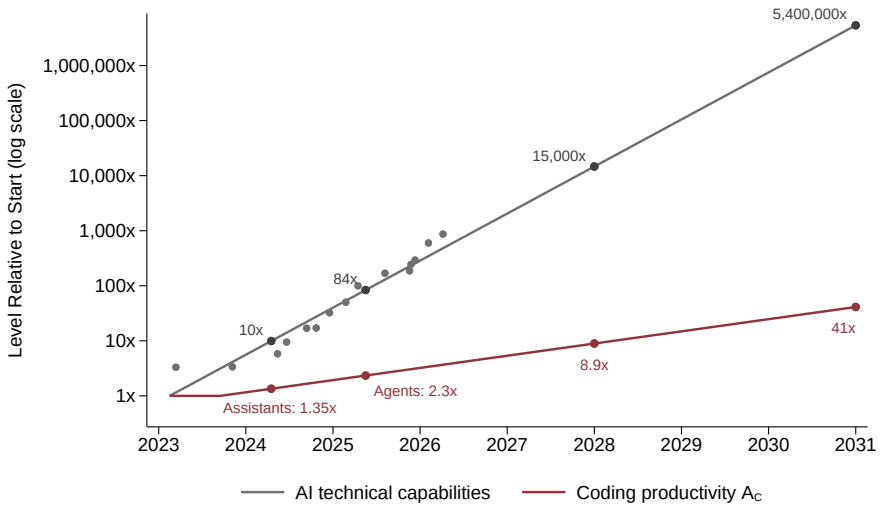
FIGURE 9. Long-run impacts of firm adjustment



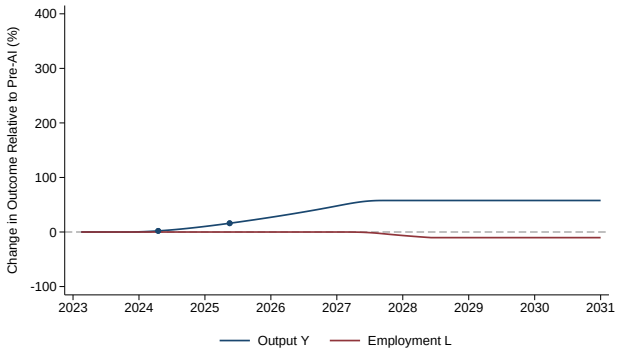
Notes: Each line is the long-run percent change in an outcome against the pre-AI adoption elasticity of product demand, evaluated at the estimated productivity changes of Table 5 and under linear demand. The dashed vertical lines indicate the thresholds above which total labor and coding labor rise. The shaded grey region indicates the range of structural estimates of the demand elasticity for software products, with their mean marked.

FIGURE 10. Projected impacts of advances in AI capabilities

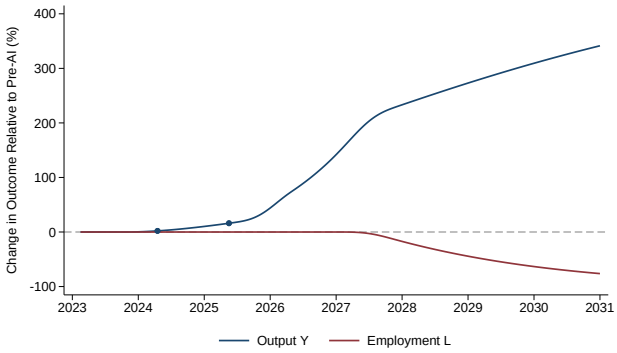
A. AI technical capabilities and coding productivity



B. Output and employment, A_R fixed



C. Output and employment, A_R grows with A_C



Notes: Panel A plots the progression of AI technical capabilities, measured by the METR 50% task-completion time horizon of frontier models (dots), and that of coding productivity A_C, each relative to its level in February 2023, on a log scale. A_C is fitted through the assistant and agent estimates and continues at the same rate per doubling of the time horizon. Panels B and C plot the model-implied changes in output and employment relative to pre-AI, at the mean demand elasticity from external sources. Labor is held fixed through 2026 and then adjusts toward its long-run level with a one-year half-life. In Panel B, review productivity A_R is held at its estimate, and output is held at its peak once labor adjusts. In Panel C, A_R grows as fast as A_C from the agent adoption date. The dots mark the measured output effects from the AI assistant and AI agent results.

B Tables

TABLE 1. Analysis sample of JF data

A. Size of analysis sample

	Number of firms	Number of workers
Analysis sample	655	720,542

B. Sample observed in data sources

	Share of firms (%)	Main unit of work activity	Number of units	Units per active worker-month
Version control data	100.0	Commits	297,586,102	24.27
		Pull requests	54,142,996	4.41
Task management data	100.0	Issues	73,139,708	5.96
Calendar data	29.3	Events	20,315,197	6.59

Notes: This table displays summary information about the analysis sample. Panel A displays the total number of firms and workers in the sample. Panel B displays the amount of coverage across the various sources of data.

TABLE 2. Firm characteristics

	Mean	Std. Dev.
Firm size		
Total workers, Jan 2023 (Revelio)	1,677	5,538
Active engineers, Jan 2023 (JF)	249	385
Firm age		
Firm age (years, as of Jan 2023)	19.4	21.6
Revenue engine		
SaaS license	0.493	
Digital service operator	0.295	
Software as input	0.136	
Customer industry		
Horizontal (business function)	0.229	
Financial services and insurance	0.156	
Healthcare and life sciences	0.111	
Other verticals	0.504	
N	655	

Notes: This table displays summary statistics for the 655 firms in our analysis sample. Firm size is measured as of January 2023: total worker counts are measured from Revelio (including non-engineering employment), and active engineer counts are the number of active workers on Jellyfish. Firm age is measured in years since each firm's Revelio-reported founding year. Revenue engine (SaaS license, digital service operator, or software as input) and customer industry are assigned using an LLM-based classification of firms' Revelio descriptions.

TABLE 3. Worker characteristics

	Share (%)
Software engineering	76.6
Data science and research	5.8
Other engineering	0.3
Product and design	7.9
Revenue	3.9
General and administrative	5.4
Chief executives	0.2

Notes: This table shows the roles of workers in our data sample. Each worker's role comes from the HR-entered job title in Jellyfish, where available, and otherwise from their matched LinkedIn job history.

TABLE 4. Balance between early versus late adopters

	AI Assistant Early Adopter (before 2024m1)	AI Agent Early Adopter (before 2025m6)
Firm size		
Log total workers, Jan 2023 (Revelio)	0.055***	0.049***
	(0.013)	(0.014)
Log active engineers, Jan 2023 (JF)	0.069***	0.089***
	(0.013)	(0.013)
Firm age		
Log firm age (years, as of Jan 2023)	0.026	-0.055**
	(0.024)	(0.025)
Revenue engine		
SaaS license	0.034	-0.012
	(0.037)	(0.039)
Digital service operator	0.009	0.115***
	(0.041)	(0.042)
Software as input	-0.025	-0.122**
	(0.054)	(0.056)
Customer industry		
Horizontal (business function)	0.023	0.035
	(0.044)	(0.046)
Financial services and insurance	-0.119**	-0.055
	(0.051)	(0.053)
Healthcare and life sciences	0.007	-0.005
	(0.059)	(0.062)
Other verticals	0.044	0.007
	(0.037)	(0.039)
N	655	655

Notes: This table assesses balance between early and late adopters of AI coding tools. Each entry reports the results from a univariate OLS regression of an early-adopter indicator on the row's firm characteristic. Column 1 defines early adoption of AI assistants as adoption before January 2024; column 2 defines early adoption of AI agents as adoption before June 2025. Firm size is measured as log total workers (Revelio) and log active engineers (JF); firm age is measured in log years since each firm's Revelio-reported founding year. Revenue engine (SaaS license, digital service operator, or software as input) and customer industry are assigned using an LLM-based classification of firms' Revelio descriptions. Standard errors in parentheses. *, **, *** indicate significance at the 10%, 5%, and 1% levels.

TABLE 5. Model parameter estimates

		Estimate
$\mathrm{d}\ln A_C$	Coding productivity	0.552***
		(0.132)
$\mathrm{d}\ln A_R$	Review productivity	-0.073
		(0.051)
$\mathrm{d}\ln L_C$	Coding labor	-0.257***
		(0.078)
$\mathrm{d}\ln L_R$	Review labor	0.361***
		(0.080)
$\mathrm{d}\ln n$	Code per unit of software output	0.159
		(0.098)
J statistic		2.914
p-value		0.233
Moments		6
Free parameters		4

* p<0.10, ** p<0.05, *** p<0.01

Notes: This table reports the parameters of the two-task production model, estimated by minimum distance from six firm-level event-study moments on the AI-agent arm. Standard errors are classical minimum-distance standard errors, which assume the covariances between the moments are zero, and take variances from the event studies. The J statistic tests the over-identifying restrictions.

TABLE 6. Sources for the elasticity of product demand

Good	ε	Source	Method
Software products: the shaded range			
Mobile apps, Google Play	3.73	Ghose and Han (2014)	BLP, top 400 apps
Mobile apps, Apple App Store	1.97	Ghose and Han (2014)	BLP, top 400 apps
Retail software	1.90	Ghose and Sundararajan (2005)	Markups inverted through Lerner
Console video games	1.50	Derdenger (2014)	Structural demand, technological tying
Range	1.50 to 3.73		
Mean	2.27		Reported headline

Notes: Estimates of the price elasticity of demand for software products, collected from the existing literature. The four estimates listed span the range shaded in Figure 9, and their mean is the headline elasticity reported in the text.

C Qualitative interviews

To assess the validity of our identification strategy and empirical interpretation, we conduct a series of qualitative interviews with software engineers, engineering managers, product managers, and related stakeholders across a range of firms. In total, we interview 21 individuals spanning different roles, seniority levels, and company types.

The interviews provide context on firms' AI adoption processes, the organization of software production, and the mechanisms through which AI coding tools affect productivity, bottlenecks, and product development. Quotes have been lightly edited for clarity.

C.1 AI adoption

Procurement Process

"We've got four teams involved in purchasing any tech. One is on the sourcing side, and handles large contracts; they understand whether the contracts are financially favorable towards [the company]. You also have legal, who will have an opinion on whether [the company's] data and IP are protected. Security gets involved as well. Finally, we expect IT to own and operate the tech centrally. The process typically takes a few weeks minimum. That can also mean months. ChatGPT came out in November 2022. Fall of 2023 is when we initially acquired a few test licenses before we rolled it out."

— Head of AI Analytics and Insights

"Our EA just asked IT and legal if it was ok to get it and got approval fast. Usually, this happens within a day for us."

— Engineering Manager

"There is an intake procurement form, and the procurement manager will go through various assessments. We go through quality assessments, cybersecurity assessments, and assessments about compliance. So on average, the approval will take anywhere between six and twelve months."

— Associate Director of Strategic Planning

Are workers allowed to use tools through individual licenses?

"Our code is our intellectual property. What happens to the code that's exposed to GitHub Copilot? ... We said, 'do not use anything that is not approved by [the company].' "

— Head of AI Analytics and Insights

"No, that is not allowed." — Engineering Manager

How was the adoption decision related to the firm's expansion plans?

"[Our company] is constantly in a growth stage, because unfortunately our shareholders care about it on a quarterly basis." — Associate Director of Strategic Planning

C.2 Effects of AI

Software production process

"We have our own product insights from seeing what customers are lacking, and then we also have direct customer requests where they say, 'hey, I need a button here that does this.' We bring all of those insights together and prioritize them using what is called a RICE score, looking at the level of effort, the impact we think it will have, and the reach — how many customers it would affect."

— Startup Founder and Former Product Manager

"Every quarter, your team does quarterly planning and decides on a bucket of projects for the quarter. You go through and assign time estimates to each task and project. Then within the quarter, every two weeks, we do sprint planning."

— Senior Software Engineer

"In an ideal team situation, around 80% of the team's effort is planned work, while 20% is focused on fixing issues, bugs, or other things that come up unexpectedly." — Startup Founder and Former Product Manager

"Work typically moves through the Jira statuses of 'Implementation', 'Review', 'QA Testing', and then 'Release'. For a task to get marked as completed, it not only has to be merged, but also deployed to production."

— Junior Software Engineer

"After the engineer finishes building the solution, it goes into the QA process and user acceptance testing. QA ensures that the new code has not broken the codebase or introduced inconsistencies, while user acceptance testing ensures that the ticket actually satisfies the intended business requirements. Once both of those are complete, the changes move into the release candidate."

— Startup Founder and Former Product Manager

"We only resolve a Jira issue once we've deployed the code associated with it to production."

— Senior Software Engineer

Coding productivity

"For simple and routine tasks, there's so much documentation, so the models are very well-trained. With Claude Code, I can write a draft in two seconds and it's going to be right 90% of the time; I just have to do a little bit of testing myself. There is a huge speed-up, on the order of 100x. But for deeper engineering tasks, it doesn't really save as much time — it gets stuff wrong a lot, and you end up spending more time in the review process." — Senior Software Engineer

"I use LLMs for nearly every task now. I do not actually type much code anymore. But I also have over a decade of software design principles, systems knowledge, and intuition." — Senior Software Engineer

"The model gets you maybe 95% of the way there, and then the remaining 5% is figuring out what it got wrong and fixing it. My work became less about writing code directly and more about reviewing and correcting what Claude produced. But it still sped up my coding work substantially."

— Junior Software Engineer

Production bottlenecks

"We were able to write code faster in some instances, but there is only one bottleneck at a time. If you speed up coding, the bottleneck shifts somewhere else. In most organizations I have worked in, the real bottleneck has been communication between business and product teams, as well as review and QA. You can generate features much faster now, but they still need to be validated and approved by humans."

— Senior Software Engineer

"People could make a huge number of commits quickly, but code review was still the bottleneck. On one project, only one senior engineer was allowed to approve changes because he was the expert on the codebase. He was so bandwidth constrained that he could only review my code once a week. After some outages that were caused by AI-generated commits, there were also new rules requiring senior engineer approval for AI-generated code, which created even more review bottlenecks."

— Junior Software Engineer

"A lot of business-value tickets are constrained by things other than coding itself. On one project, I spent

six months waiting for legal approval for something that probably should have taken a month. Coding agents can dramatically increase coding productivity, but Jira tickets often capture bottlenecks that AI cannot solve."

— Senior Software Engineer

Review process

"Now, pull requests are getting bigger and harder to review. It is honestly becoming more anxiety inducing to even open the pull request and begin the review."

— Senior Software Engineer

"Even with AI tools, you absolutely still need humans involved in the review process. I don't see a time where you will ever not need human oversight, even as tools get better. The AI tools lack the comprehension of the overview of the entire feature, and what it is supposed to do."

— Senior Software Engineer

"It's typically our senior engineers who do reviews. It's about having ownership over the code and the impacts of shipping buggy code."

— Senior Software Engineer

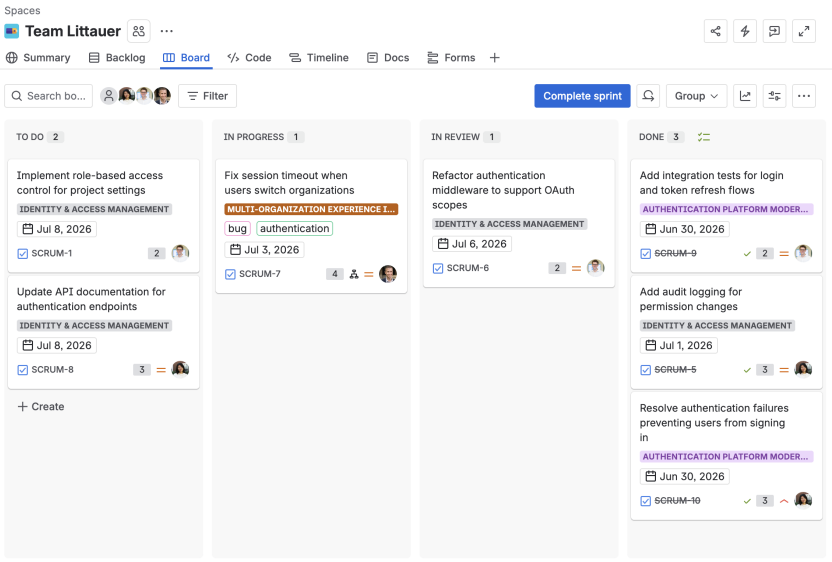
"When teams assign code review, assignment usually boils down to availability. But, if a PR or area of code is complex, I expect at least one reviewer to be senior."

— Senior Software Engineer

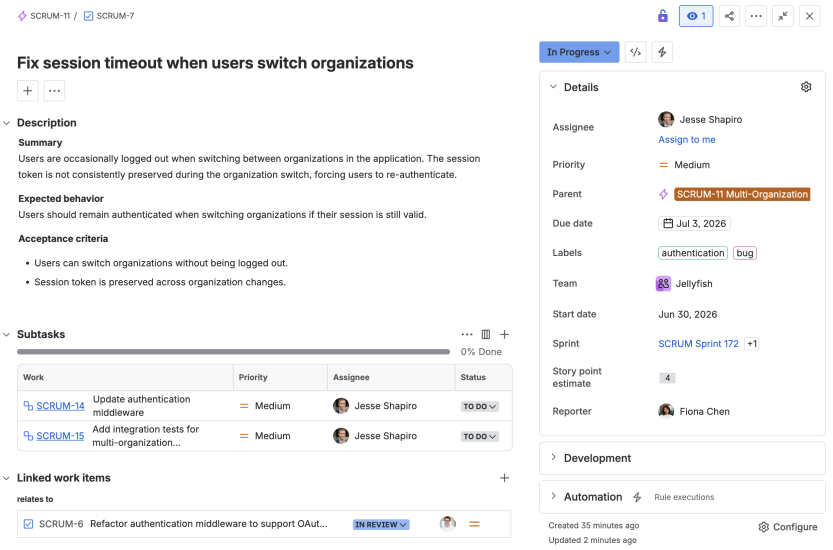
D Appendix Figures

FIGURE A1. Examples of Jira interface

A. Jira board



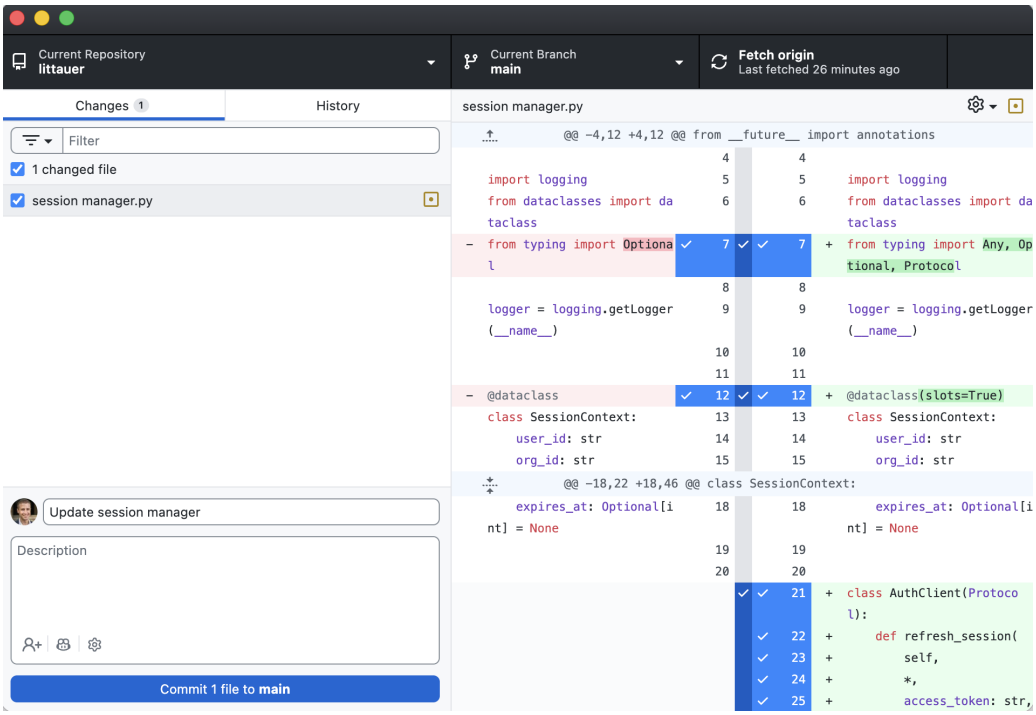
B. Jira issue



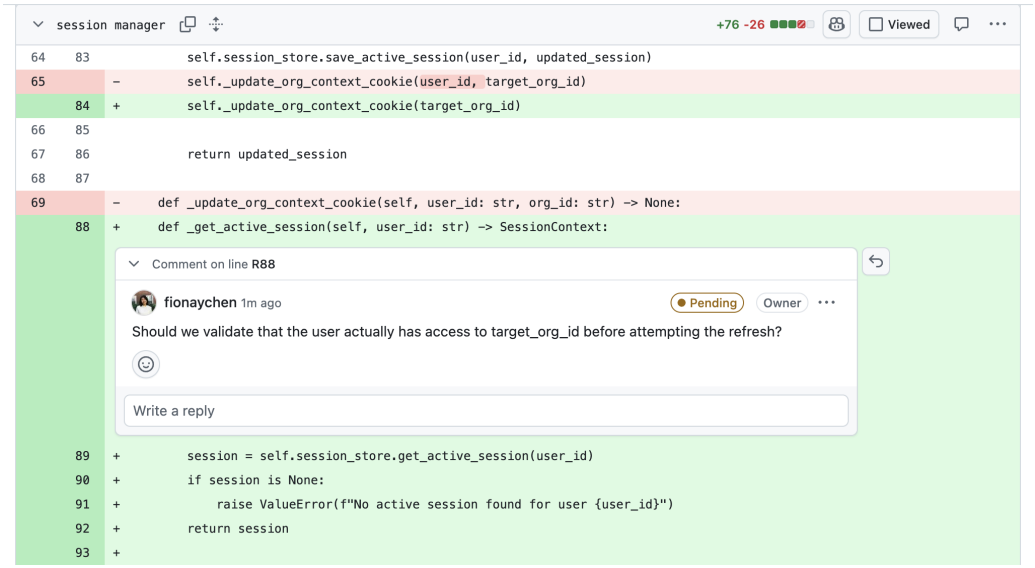
Notes: This figure displays two examples of the Jira interface. Panel A shows a team's Jira board, tracking the progression of issues throughout the different stages of software development. Panel B shows the interface for an individual Jira issue, including its description, subtasks, linked epic, linked work items, and assignee.

FIGURE A2. Examples of GitHub interface

A. GitHub commit



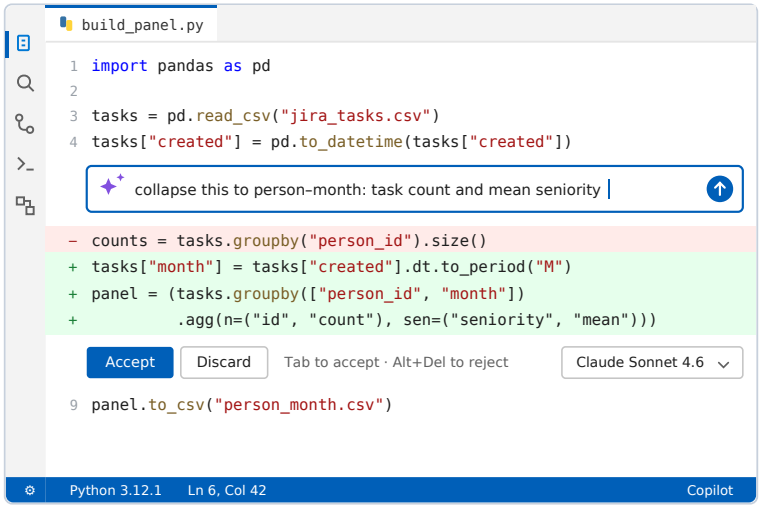
B. GitHub pull request review



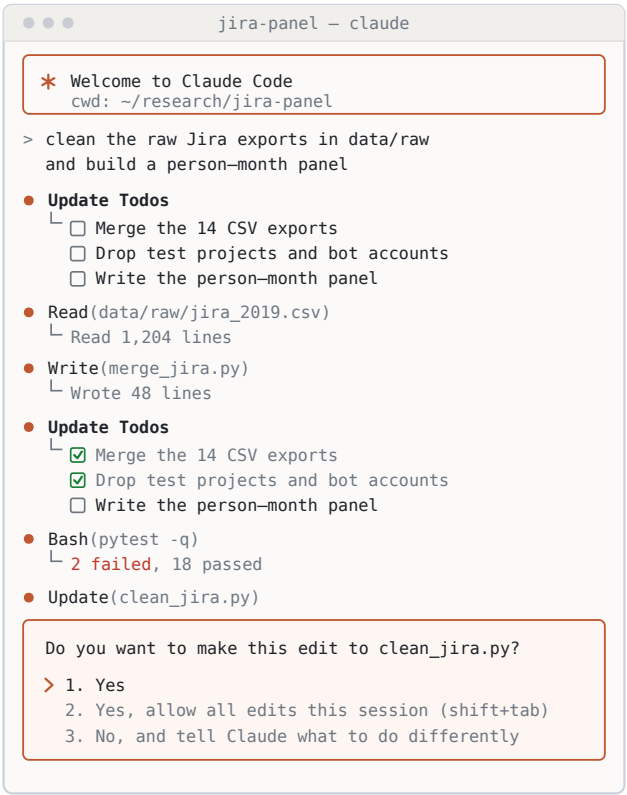
Notes: This figure displays two examples of the GitHub interface. Panel A shows the interface for creating a commit, including the modified files, code changes, and commit message. Panel B shows the pull request review interface, where collaborators review proposed code changes and provide inline comments before the changes are merged.

FIGURE A3. Examples of AI coding tool interface

A. AI coding assistant

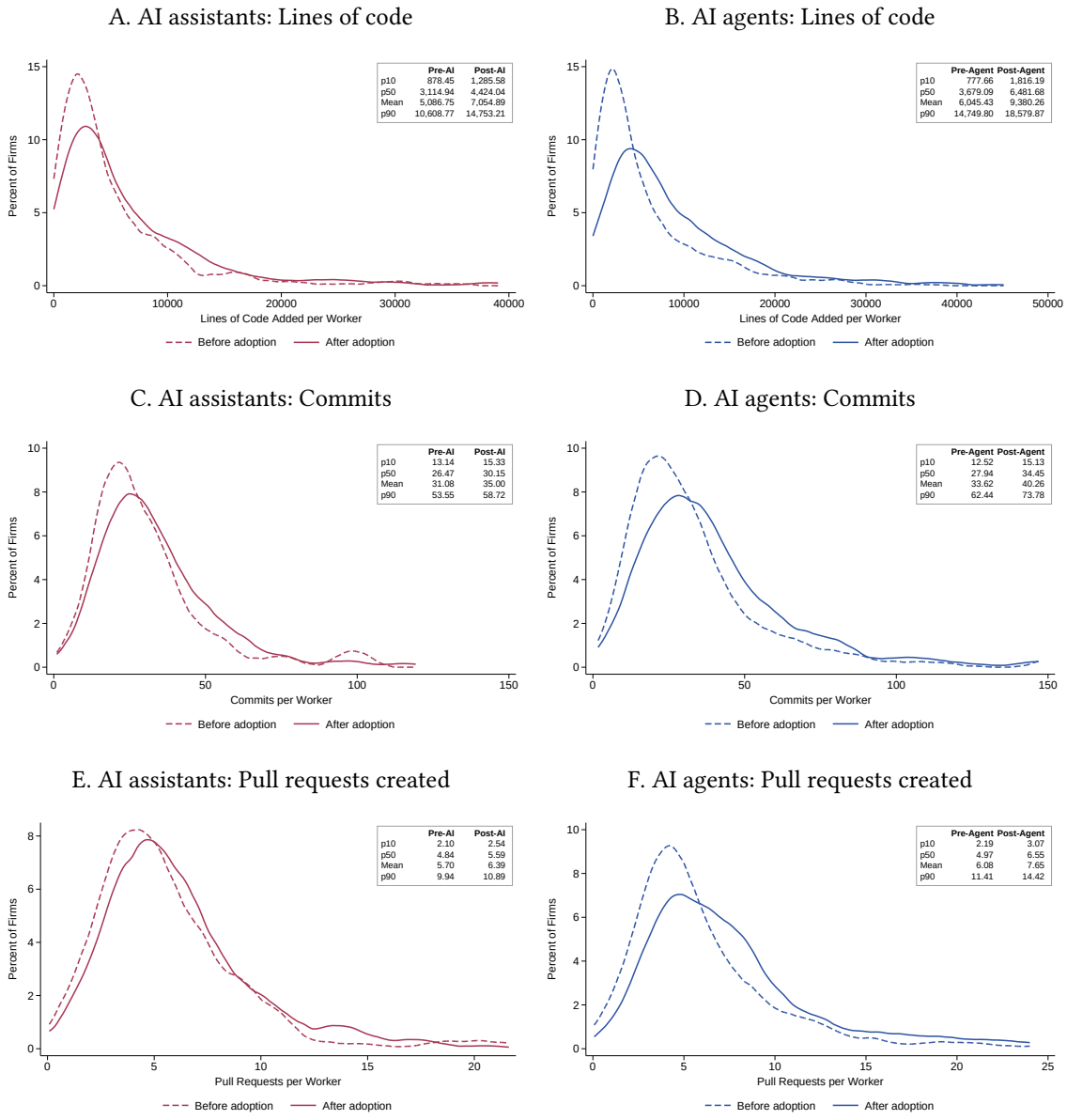


B. AI coding agent



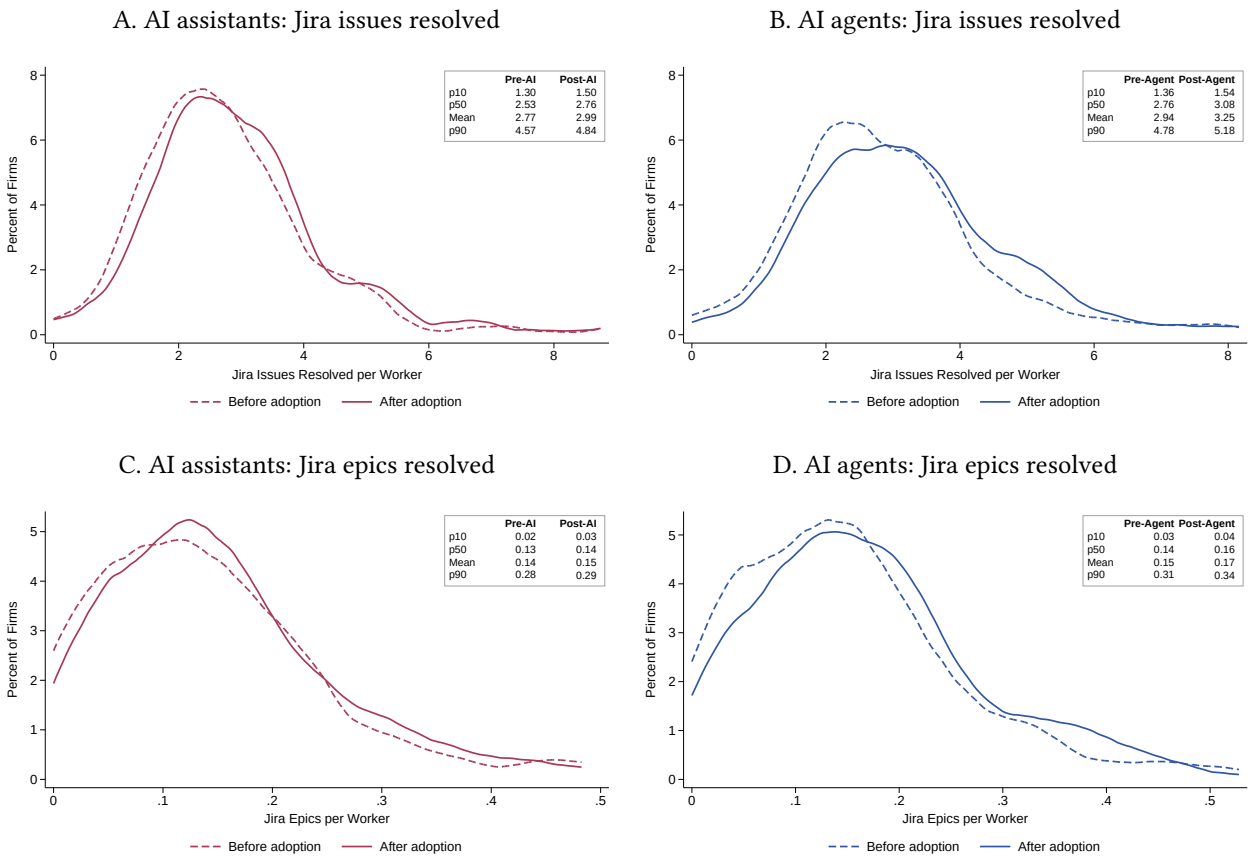
Notes: This figure displays two examples of the user interfaces for AI coding tools. Panel A displays the interface for an AI coding assistant, GitHub Copilot. Here, the user prompts the assistant, and the AI tool generates an individual block of code. Panel B displays the interface for an AI coding agent, Claude Code. Here, the user provides a longer prompt, and the AI tool generates a plan as well as an entire file of code.

FIGURE A4. Distributions, pre- and post-AI adoption: code output



Notes: This figure displays the raw firm-level distributions of the code output measures in Figure 3, before and after adoption. Each adopting firm contributes its mean over the eight months before adoption (dashed line) and its mean over the months after adoption (solid line): months 1 to 16 for AI assistants and months 1 to 12 for AI agents. Each firm that never adopts contributes its mean over the estimation window (gray line). Curves are kernel densities, in percent of firms, and the horizontal axis is cut at the 99th percentile of the firm means. In Panels A, C, and E, the treatment is adoption of AI assistants. In Panels B, D, and F, the treatment is adoption of AI coding agents.

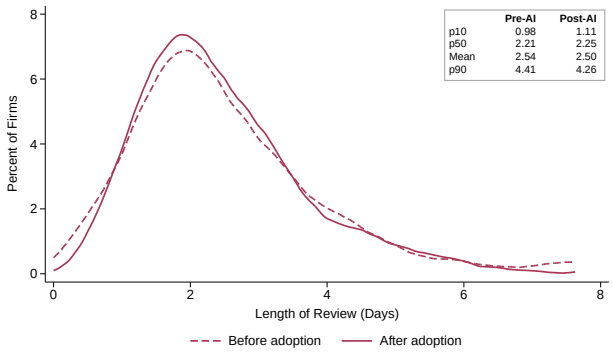
FIGURE A5. Descriptives: software output



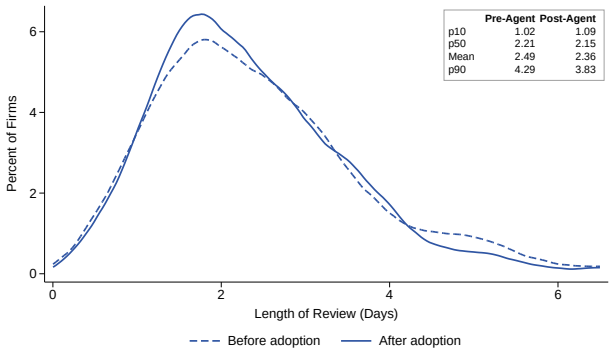
Notes: This figure displays the raw firm-level distributions of the software output measures in Figure 4, Jira issues (tasks and stories) and Jira epics resolved per worker, before and after adoption. Months before the start of the firm's Jira coverage are dropped. Each adopting firm contributes its mean over the eight months before adoption (dashed line) and its mean over the months after adoption (solid line): months 1 to 16 for AI assistants and months 1 to 12 for AI agents. Each firm that never adopts contributes its mean over the estimation window (gray line). Curves are kernel densities, in percent of firms, and the horizontal axis is cut at the 99th percentile of the firm means. In Panels A and C, the treatment is adoption of AI assistants. In Panels B and D, the treatment is adoption of AI coding agents.

FIGURE A6. Descriptives: code review

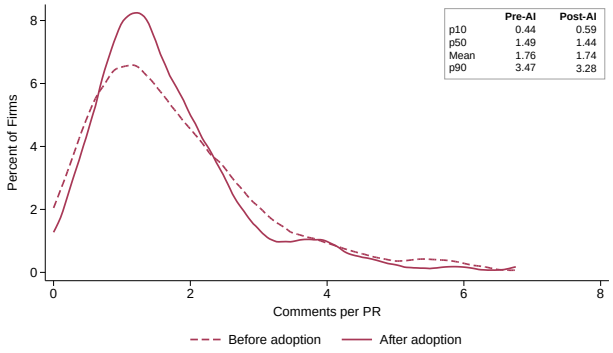
A. AI assistants: Days from PR open to merge



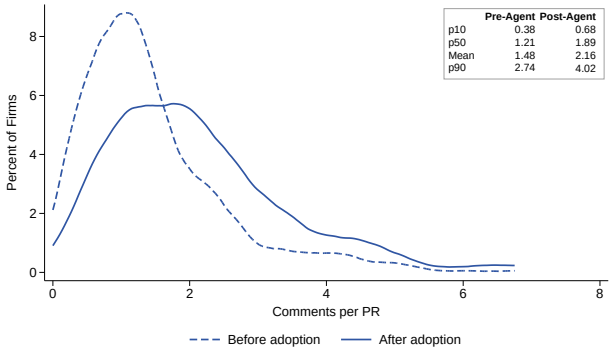
B. AI agents: Days from PR open to merge



C. AI assistants: Comments per pull request

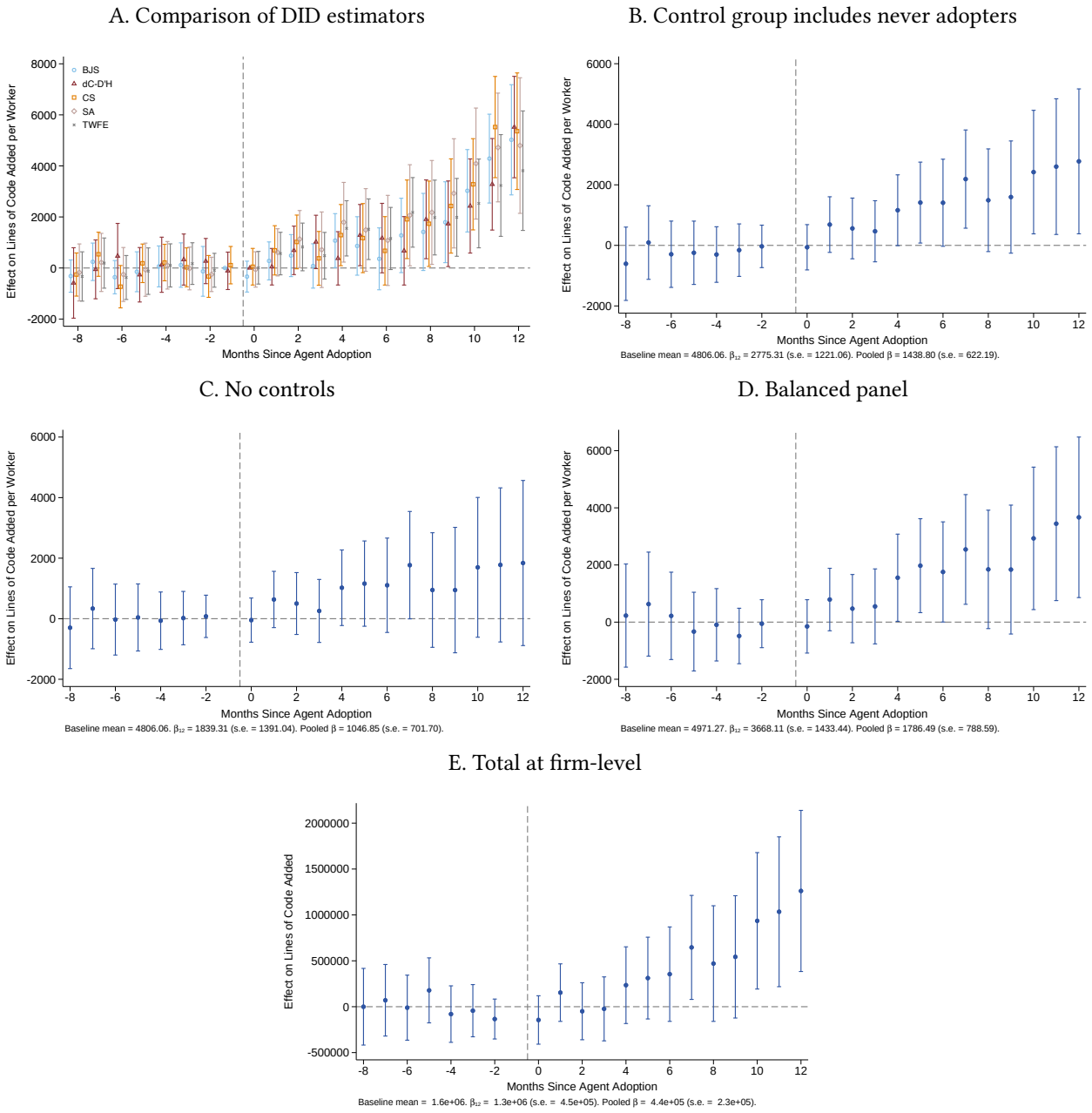


D. AI agents: Comments per pull request



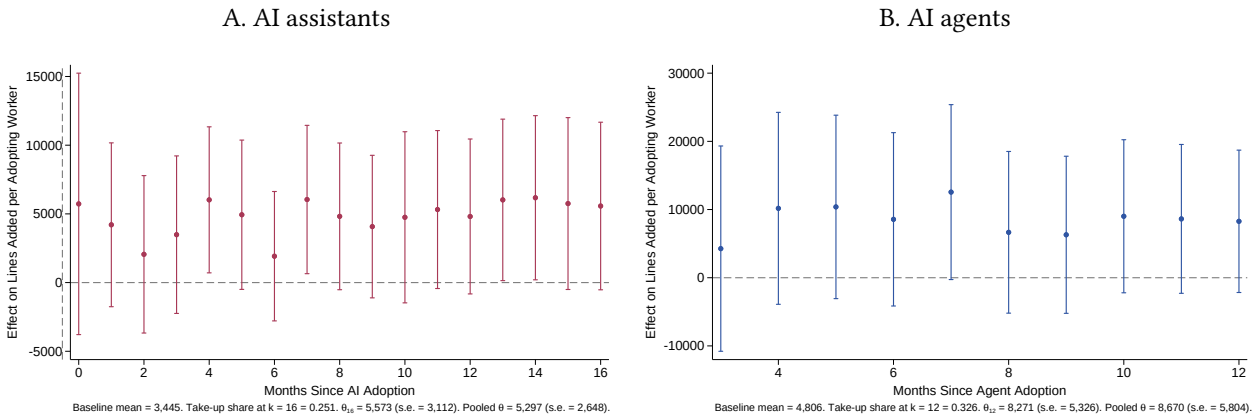
Notes: This figure displays the raw firm-level distributions of two code review measures, days from PR open to merge and comments per pull request, before and after adoption. Each adopting firm contributes its mean over the eight months before adoption (dashed line) and its mean over the months after adoption (solid line): months 1 to 16 for AI assistants and months 1 to 12 for AI agents. Each firm that never adopts contributes its mean over the estimation window (gray line). Curves are kernel densities, in percent of firms, and the horizontal axis is cut at the 99th percentile of the firm means. In Panels A and C, the treatment is adoption of AI assistants. In Panels B and D, the treatment is adoption of AI coding agents.

FIGURE A7. Alternative specifications



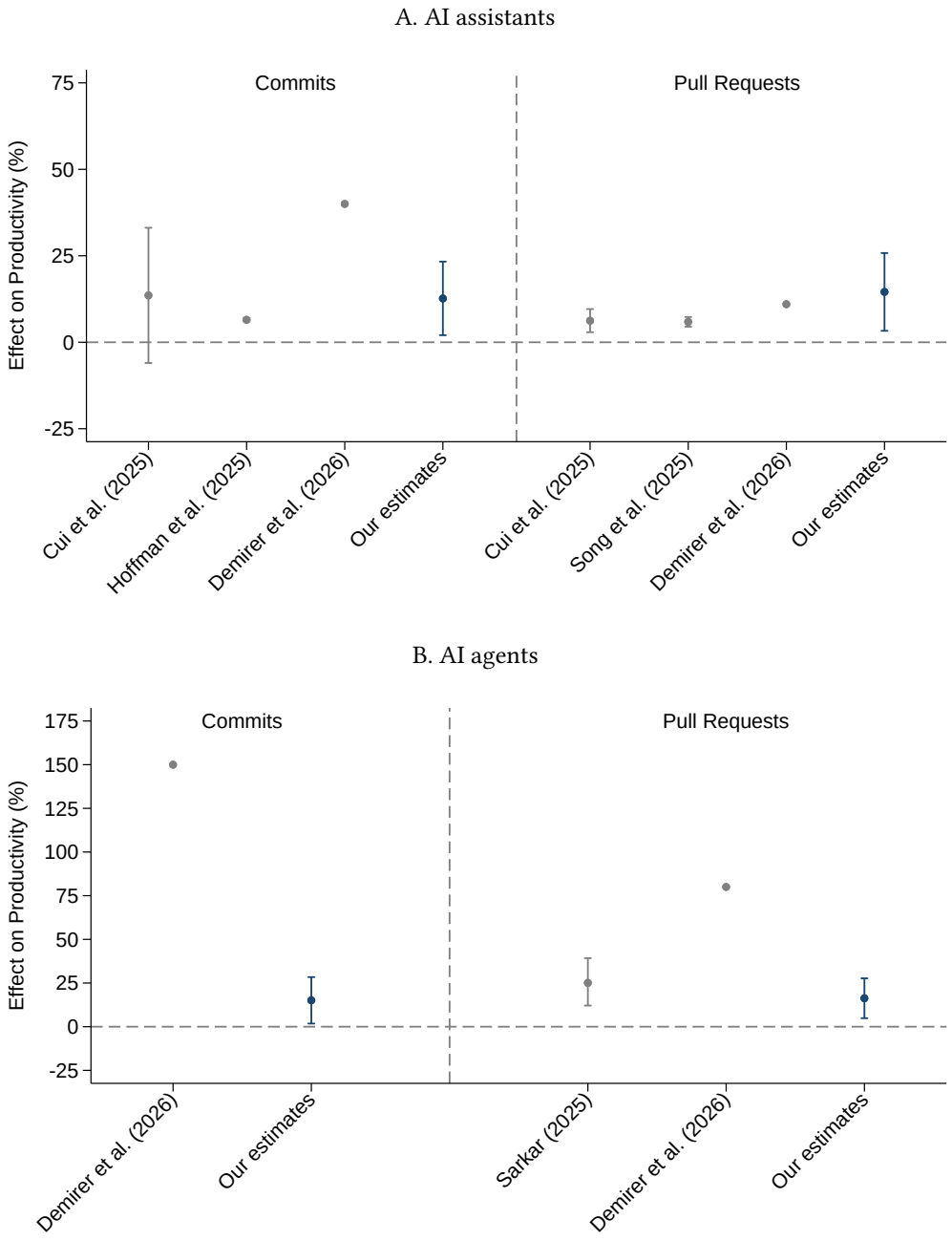
Notes: This figure displays the effects of agent adoption on lines of code added per worker-month under alternative specifications. Each panel changes one feature of the main specification. Panel A compares two-way fixed effects with the estimators of Borusyak et al. (2024), de Chaisemartin & D'Haultfœuille (2024), Callaway & Sant'Anna (2021), and Sun & Abraham (2021). Panel B adds the firms that never adopt AI to the comparison group. Panel C estimates the event study without the time-varying controls for firm characteristics. Panel D re-estimates the specification on a balanced panel of treated firms, defined as having data from 8 months before to 12 months after adoption. Never adopters remain in the comparison group, since without them the late calendar months have no untreated firms. Panel E uses total lines of code added at the firm level, rather than normalized per worker.

FIGURE A8. IV estimates: effect per adopting worker on lines of code added



Notes: This figure displays two-stage least squares estimates of the effect of individual take-up on lines of code added per worker-month. The endogenous variables are the share of the firm's active workers who have personally adopted AI, interacted with an indicator for the month since firm-level adoption. The instruments are the corresponding indicators for months since firm-level adoption. Thus, each plotted coefficient is the effect per worker who has taken up the tool. The regressions include the same fixed effects and controls as the main specification, and standard errors are clustered by firm.

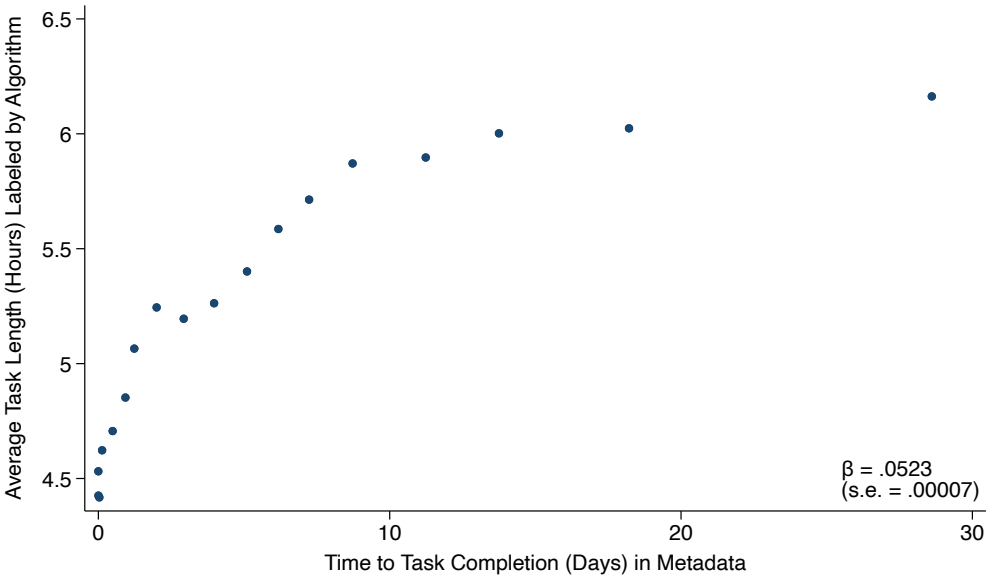
FIGURE A9. Comparison of productivity estimates



Notes: This figure displays a comparison of estimates of AI coding tools' productivity impacts, taking the pooled estimates from our paper and comparing to those of Cui et al. (2024); Hoffmann et al. (2024); Song et al. (2024); Sarkar (2025), and Demirer, Musolff, & Yang (2026), which are the papers that similarly report effects on commits and pull requests.

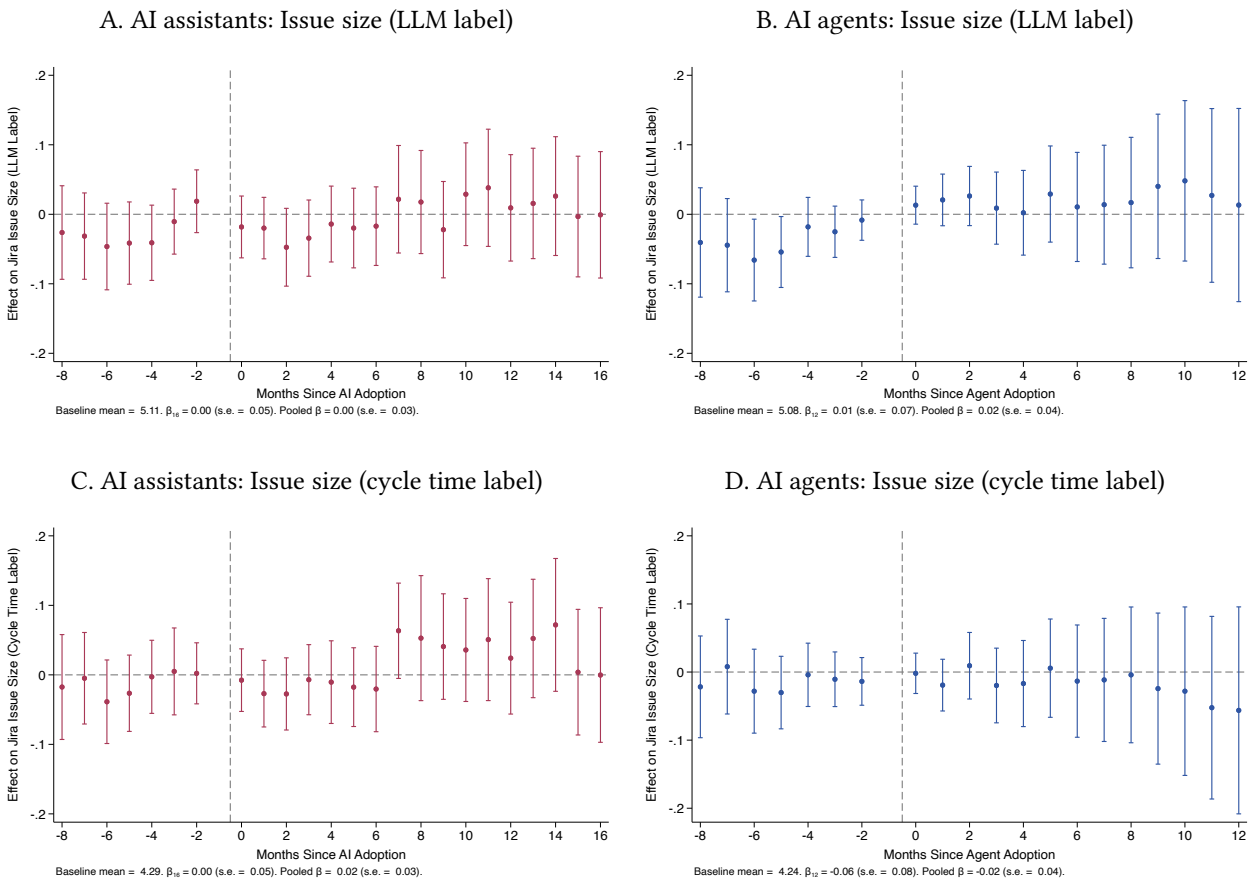
FIGURE A10. Relationship between LLM label and cycle time

A. Comparison of algorithm-assigned "length" label to task completion time



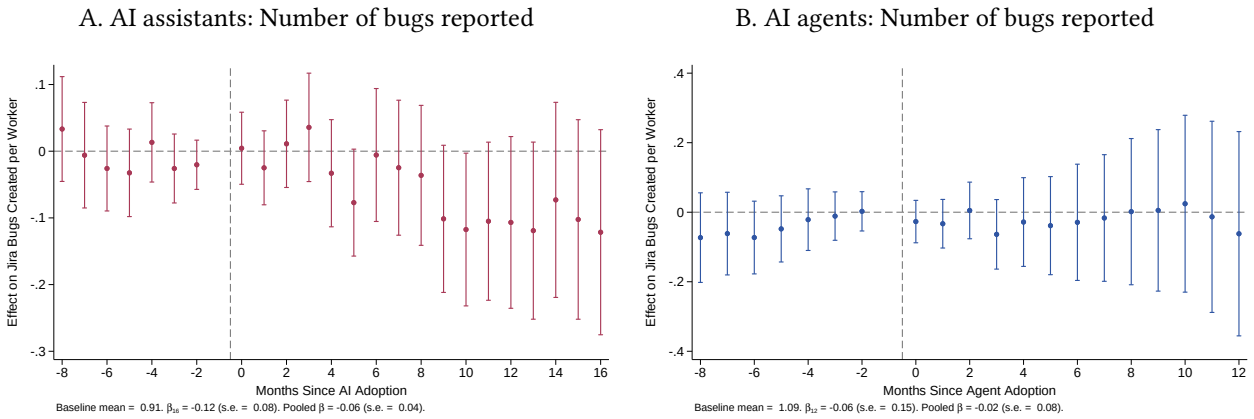
Notes: This figure shows the validation of the "length" label assigned by our machine learning algorithm against days to task completion, as measured in Jira. We winsorize task completion time at 30 days.

FIGURE A11. Effects of AI on Jira issue size



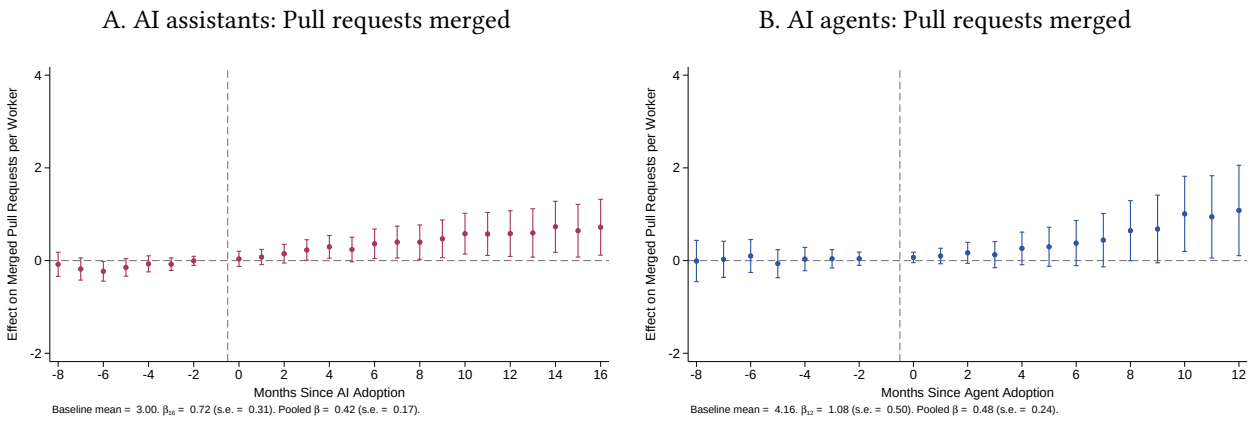
Notes: This figure displays event studies of the effects of AI adoption on the average size of Jira issues. In Panels A and B, issue size is measured using the LLM-based length label described in Section F.2. In Panels C and D, issue size is measured using the predicted log cycle time from the same architecture trained on pre-period data. In Panels A and C, the treatment is adoption of AI assistants. In Panels B and D, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A12. Effects of AI on software output quality



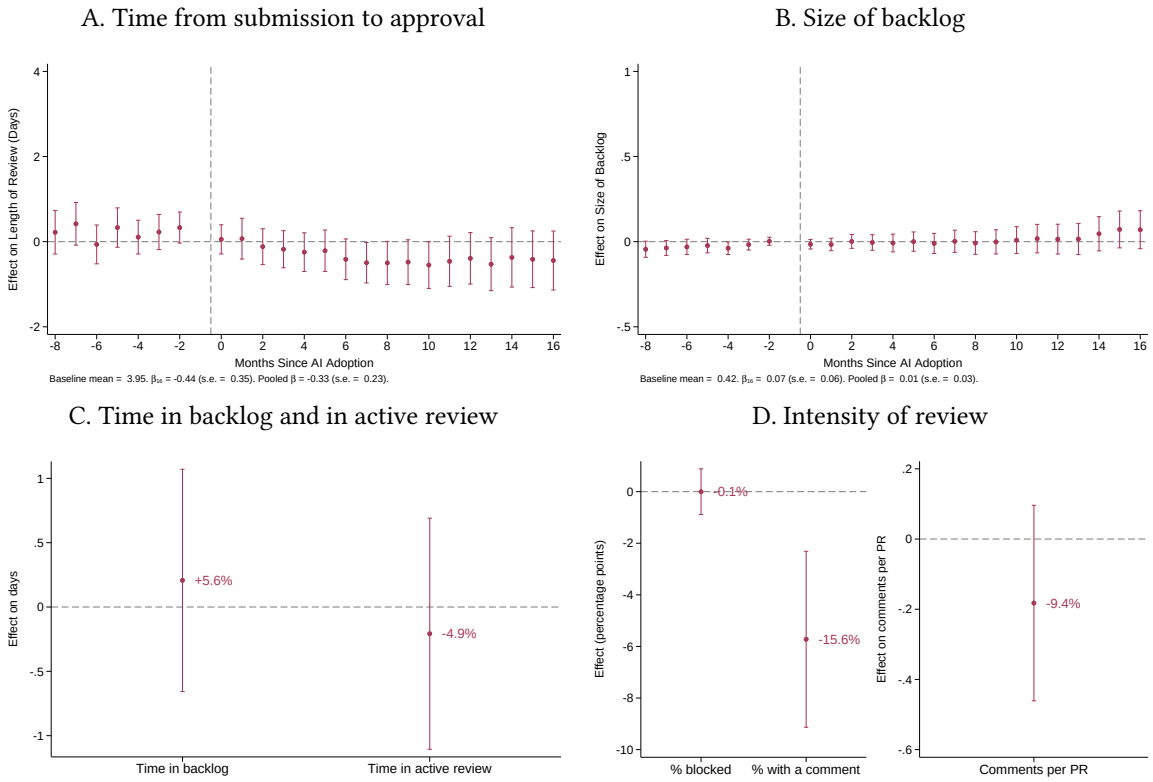
Notes: This figure displays event studies of the effects of AI adoption on the quality of software output. In Panels A and B, the outcome variable is the number of bugs reported in Jira, normalized by the number of workers. We construct a firm-by-month panel, where the outcome variable in each row is the Jira bugs created per worker-month — i.e., the total number of Jira bugs created in the firm that month, divided by the number of engineers active in JF in that firm-month. In Panel A, the treatment is adoption of AI assistants. In Panel B, the treatment is adoption of AI agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A13. Effects of AI on pull requests merged



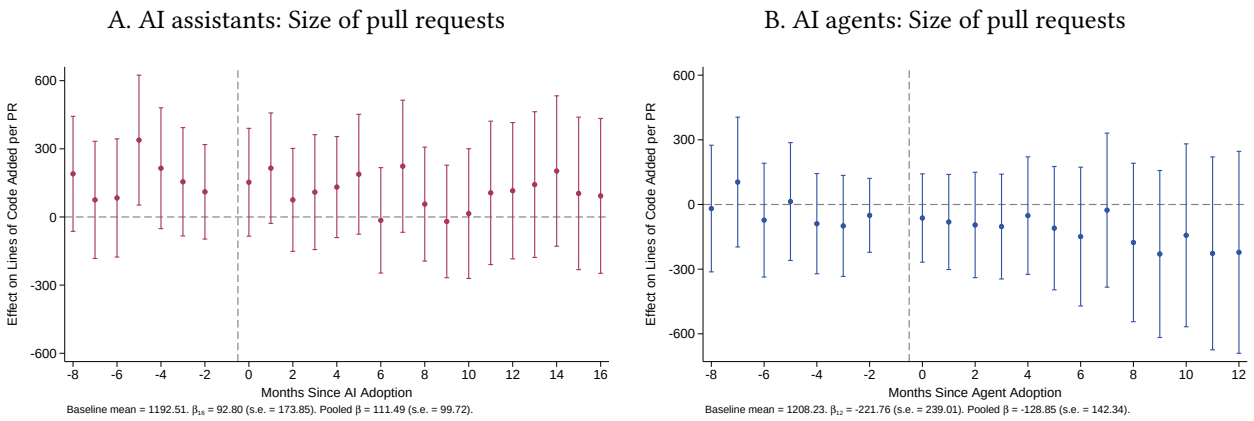
Notes: This figure displays event studies of the effects of AI adoption on the number of pull requests merged. In Panels A and B, the outcome variable is the number of pull requests merged, normalized by the number of workers — i.e., the total number of pull requests merged by the firm that month, divided by the number of engineers active in JF in that firm-month. In Panel A, the treatment is adoption of AI assistants. In Panel B, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A14. Effects of AI assistants on code review



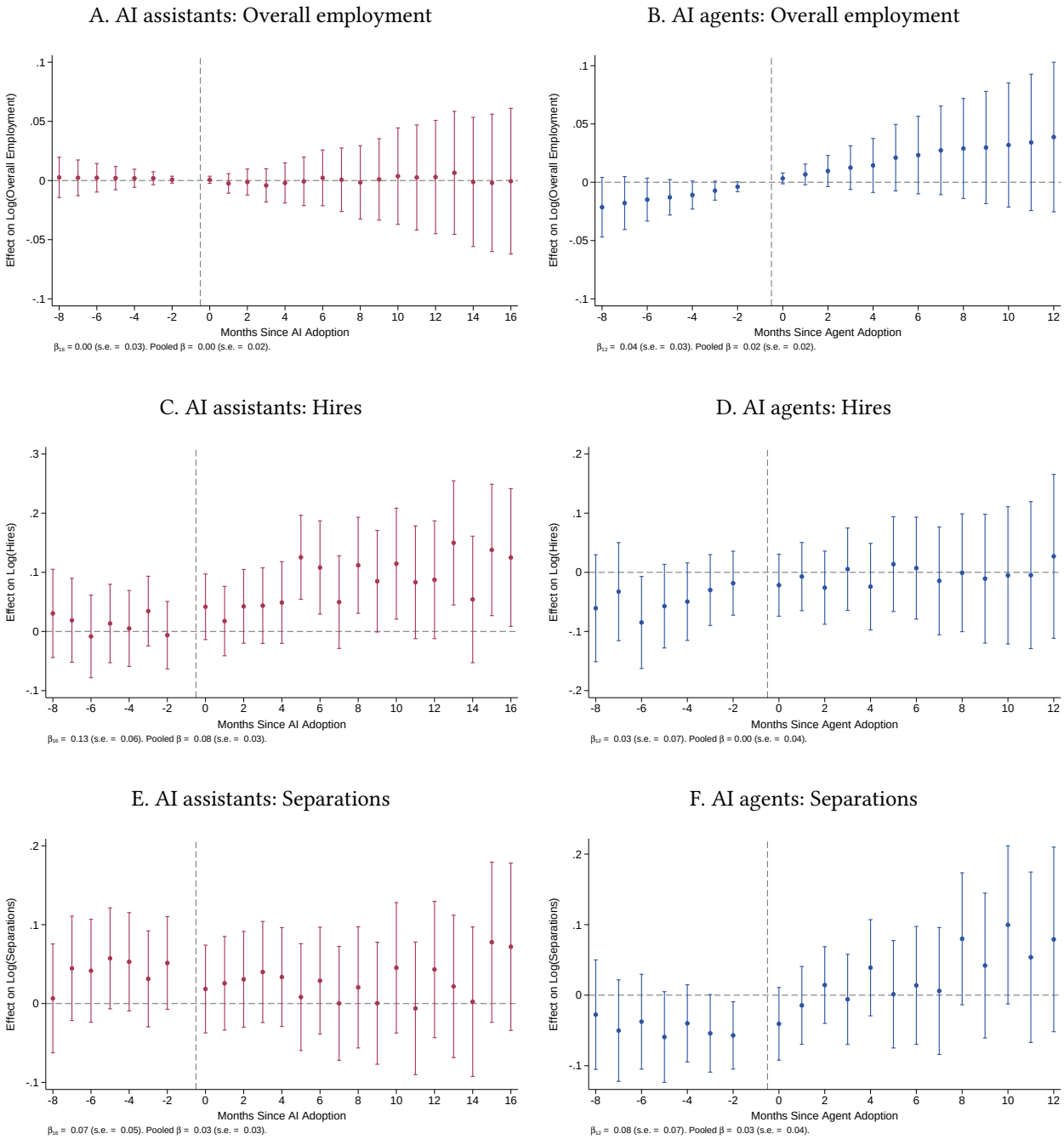
Notes: This figure displays the effects of AI assistant adoption on the code review process. Panel A displays an event study of the length of review, the number of days between when a pull request is submitted and when it is merged. Panel B displays an event study of the size of the backlog, the number of PRs open at the firm when a new PR is added. Panels C and D display $\hat{\beta}_{16}$, with its 95% confidence interval; the label beside each point gives $\hat{\beta}_{16}$ as a percent of the outcome's pre-adoption mean. In Panel C, review length is split into time in backlog, the number of days from when a pull request is submitted to its first comment from a human other than its author, and time in active review, the number of days from that first comment to merge. In Panel D, the left half displays the share of pull requests for which at least one reviewer submitted a formal request for changes, and the share of pull requests that received at least one comment from someone other than the author, both in percentage points; the right half displays the number of review comments per pull request. In all panels, the treatment is adoption of AI assistants. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A15. Effects on pull request size



Notes: This figure displays event studies of the effects of AI adoption on the average size of pull requests. In Panels A and B, the outcome variable is the number of lines of code added in each pull request. In Panel A, the treatment is adoption of AI assistants. In Panel B, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

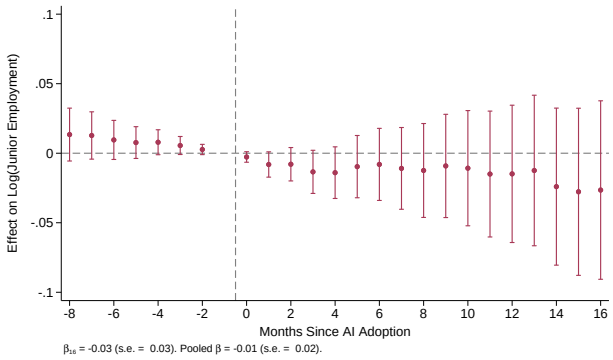
FIGURE A16. Effects of AI on overall firm employment, hires, and separations, from Revelio



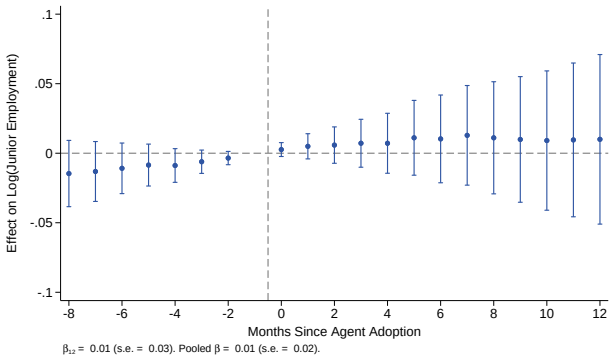
Notes: This figure displays event studies of the effects of AI adoption on log employment. In Panels A and B, the outcome variable is the overall employment, measured as the number of LinkedIn profiles attached to each firm each month (not restricting to engineers). In Panels C and D, the outcome variable is the number of external hires, measured as the number of LinkedIn profiles starting their first position at the firm each month. In Panels E and F, the outcome variable is the number of separations, measured as the number of LinkedIn profiles ending their last position at the firm each month. Hires and separations enter as $\log(1 + x)$. In Panels A, C, and E, the treatment is adoption of AI assistants. In Panels B, D, and F, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A17. Effects of AI on employment, by worker seniority

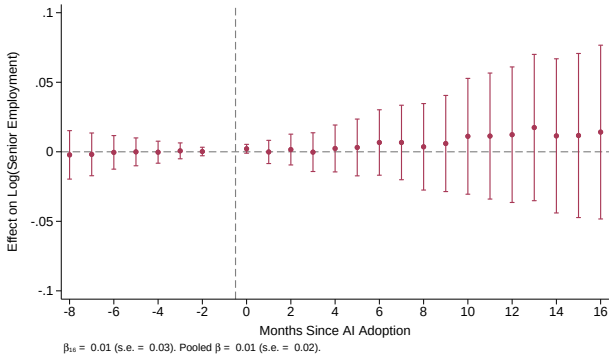
A. AI assistants: Overall junior employment (from Revelio)



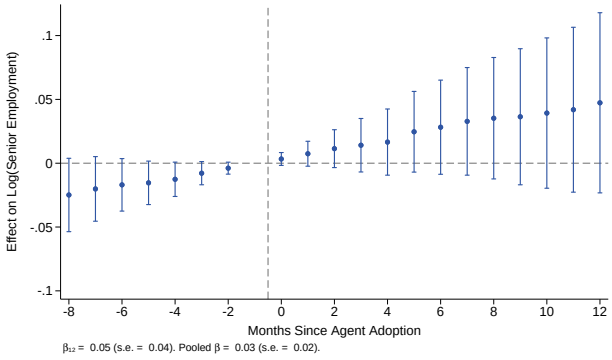
B. AI agents: Overall junior employment (from Revelio)



C. AI assistants: Overall senior employment (from Revelio)

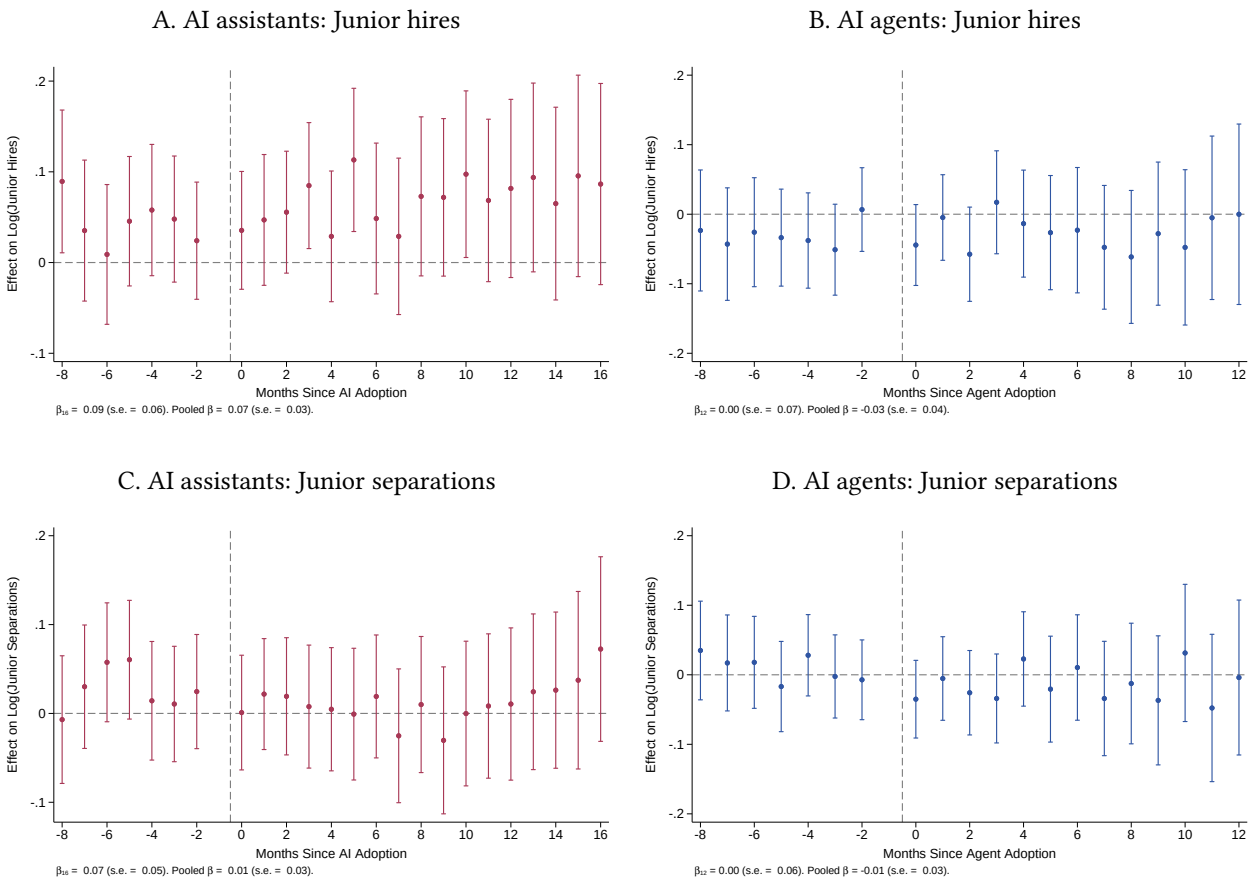


D. AI agents: Overall senior employment (from Revelio)



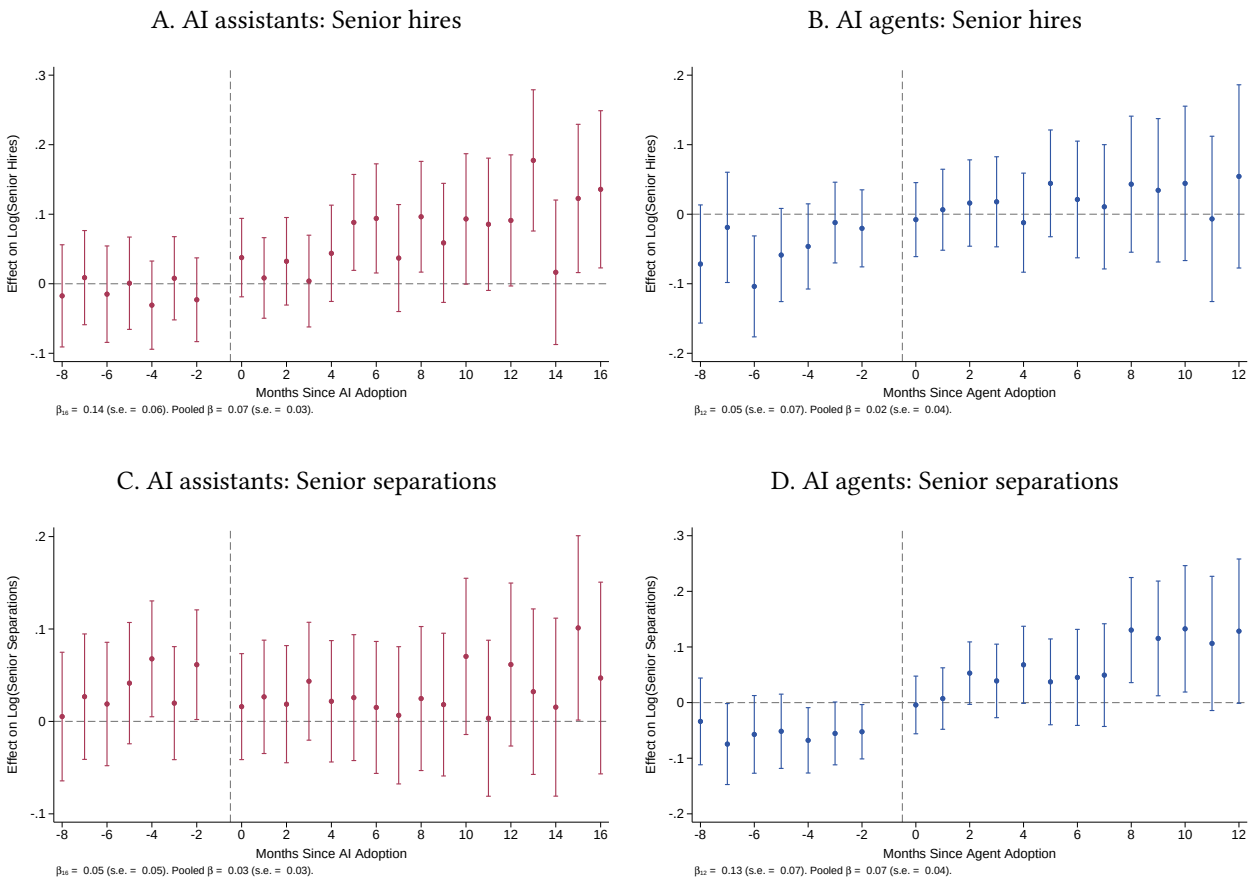
Notes: This figure displays event studies of the effects of AI adoption on log employment, separately by junior and senior workers. In Panels A and B, the outcome variable is the number of LinkedIn profiles of junior workers attached to the firm each month. In Panels C and D, the outcome variable is the number of LinkedIn profiles of senior workers attached to the firm each month. In Panels A and C, the treatment is adoption of AI assistants. In Panels B and D, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A18. Effects of AI on hires and separations of junior workers



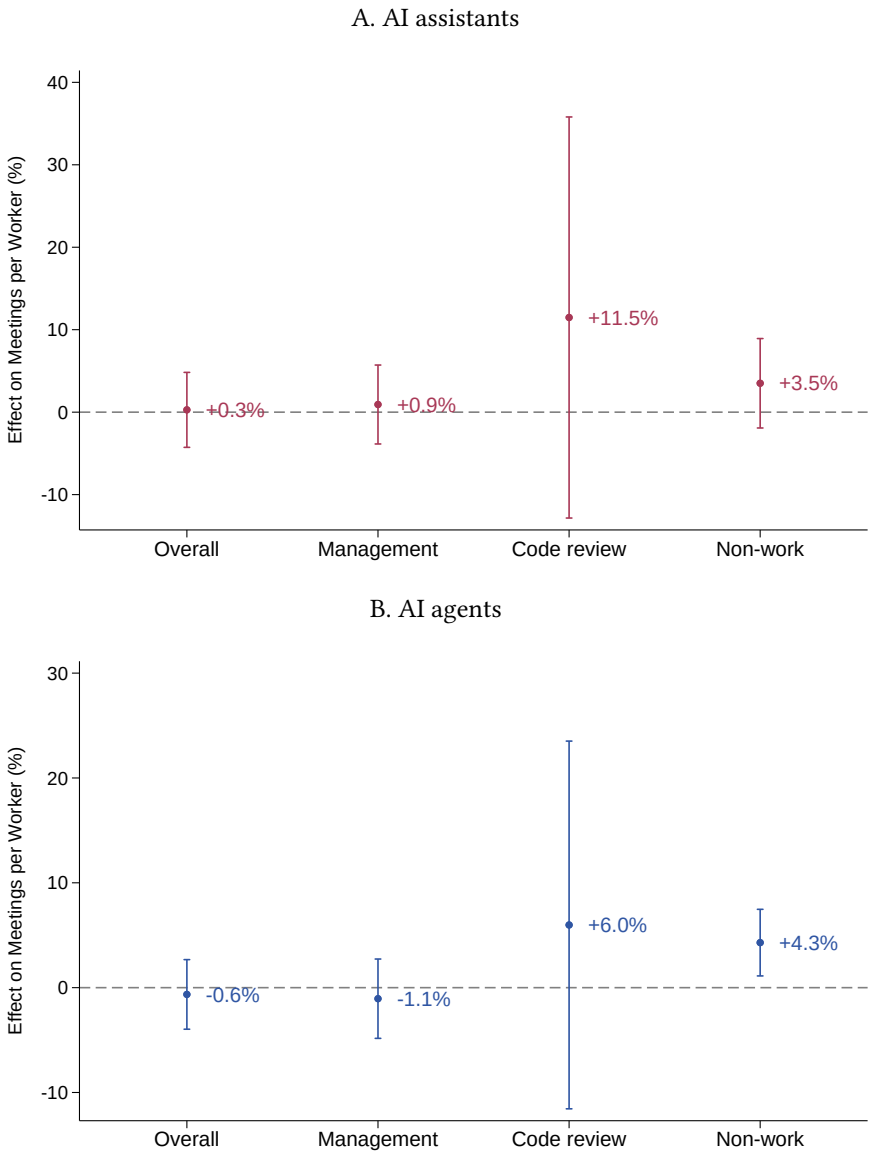
Notes: This figure displays event studies of the effects of AI adoption on hires and separations of junior workers, from Revelio. In Panels A and B, the outcome variable is the number of external hires, measured as the number of LinkedIn profiles starting their first position at the firm each month. In Panels C and D, the outcome variable is the number of separations, measured as the number of LinkedIn profiles ending their last position at the firm each month. Junior workers are those at Revelio seniority levels 1–2, taken from the first position for hires and the last position for separations. Hires and separations enter as $\log(1 + x)$. In Panels A and C, the treatment is adoption of AI assistants. In Panels B and D, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A19. Effects of AI on hires and separations of senior workers



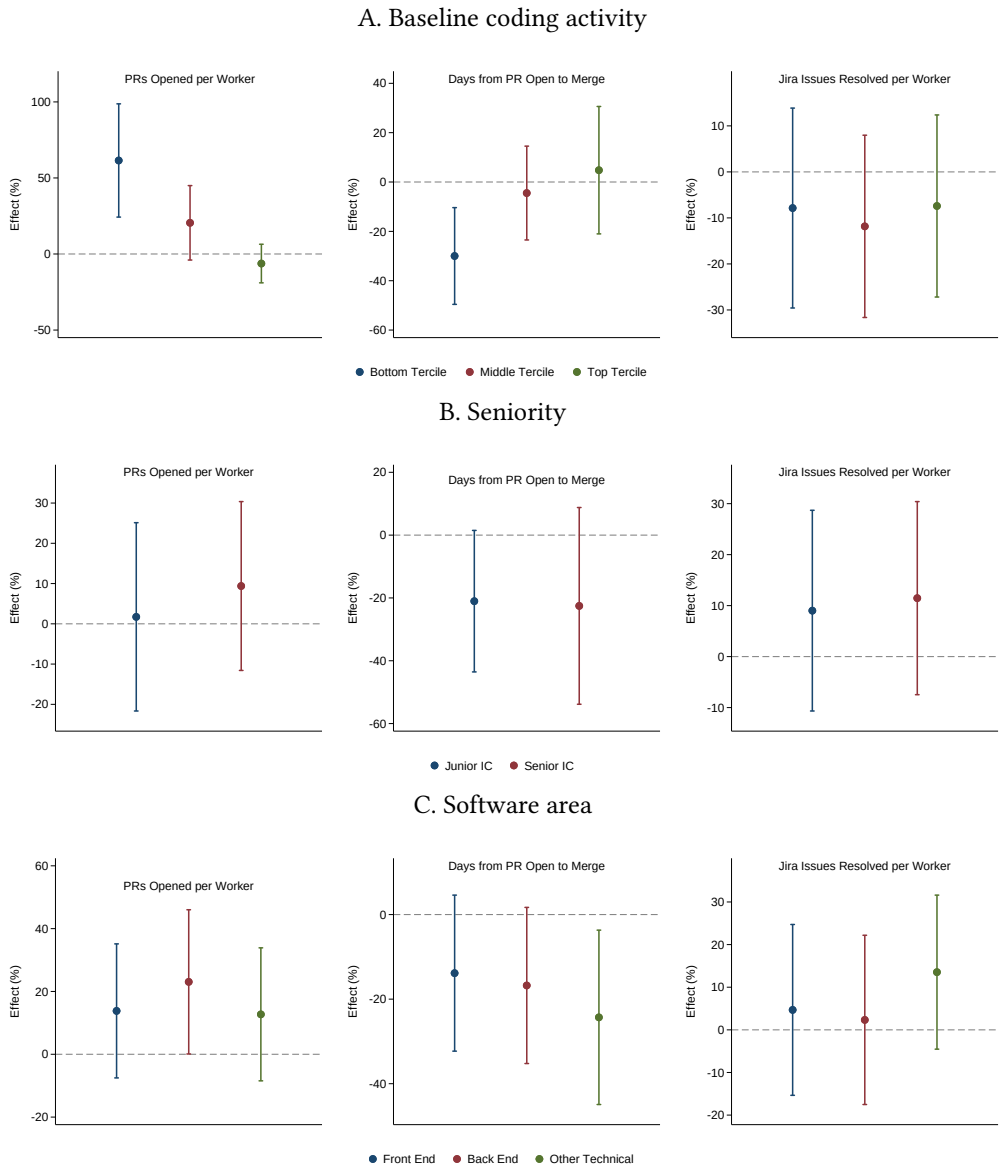
Notes: This figure displays event studies of the effects of AI adoption on hires and separations of senior workers, from Revelio. In Panels A and B, the outcome variable is the number of external hires, measured as the number of LinkedIn profiles starting their first position at the firm each month. In Panels C and D, the outcome variable is the number of separations, measured as the number of LinkedIn profiles ending their last position at the firm each month. Senior workers are those at Revelio seniority levels 3–7, taken from the first position for hires and the last position for separations. Hires and separations enter as $\log(1 + x)$. In Panels A and C, the treatment is adoption of AI assistants. In Panels B and D, the treatment is adoption of AI coding agents. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE A20. Effects of AI on calendar activity



Notes: This figure displays the effects of AI adoption on calendar activity, from Google Calendar. Each point is a difference-in-differences estimate that compares months 0 to 16 after adoption of AI assistants (Panel A), or months 0 to 12 after adoption of AI agents (Panel B), with the eight months before adoption, as a percentage of the pre-adoption mean, with 95% confidence intervals clustered by firm. Months more than eight before or more than sixteen (Panel A) or twelve (Panel B) after adoption enter as separate indicators. For each worker and month, we count the calendar events the worker accepted. We winsorize each worker-month count at the 99th percentile and take the mean across the firm's calendar users. "Overall" counts all events that JF classifies as work. "Management" counts work events other than review — i.e., planning meetings, standups and syncs, one-on-ones, and other meetings. "Code review" counts events whose title refers to code review, pull requests, or merge requests, or whose description refers to code or PR review. "Non-work" counts events that JF does not classify as work, which are mostly location markers, out-of-office blocks, and lunch. Each regression includes firm and month fixed effects, baseline engineering employment interacted with month, and binned months since the firm's GitHub coverage began.

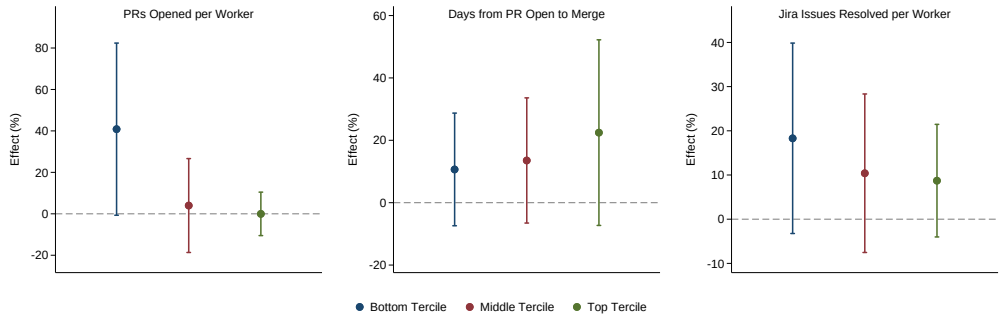
FIGURE A21. Heterogeneity by worker characteristics - AI assistants



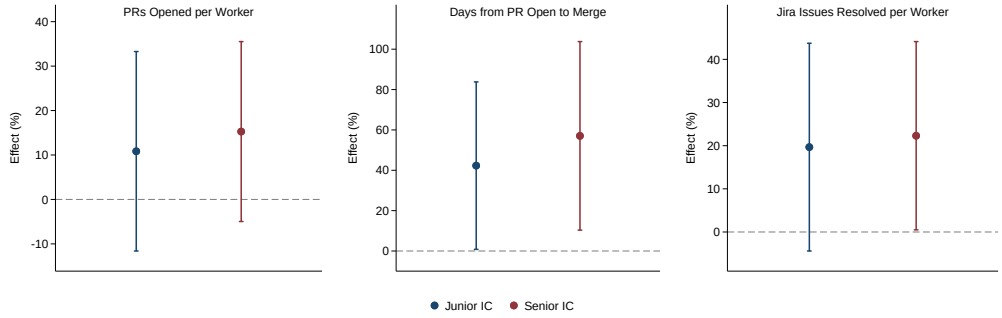
Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding assistants, by worker characteristics. Each point is the event-study coefficient sixteen months after adoption for one group of workers, as a percentage of that group's pre-adoption mean, with 95% confidence intervals clustered by firm. We construct a firm-by-worker group-by-month panel, assigning each worker to a single group for the whole sample period. PRs opened and Jira issues (tasks and stories) resolved per worker divide the group's total by the number of its workers active in JF that firm-month, winsorized at the 99th percentile. Days from PR open to merge is the mean over the group's merged pull requests. Each regression interacts the event-time indicators with the worker group, with firm-by-group and month fixed effects, baseline engineering employment interacted with month, and binned months since the firm's GitHub or Jira coverage began. The sample is the firms that adopt AI coding assistants during our sample period. In Panel A, workers are split into terciles of pull requests opened in the baseline period (January to June 2022), among workers who opened at least one. In Panel B, software engineers are classified by job title as junior individual contributors, senior individual contributors, or engineering managers. In Panel C, workers are assigned the area, based on keywords in repository names, that accounts for most of the pull requests they opened before their firm adopted; other technical includes infrastructure, data, and mobile, and workers with non-technical job titles are excluded.

FIGURE A22. Heterogeneity by worker characteristics - AI agents

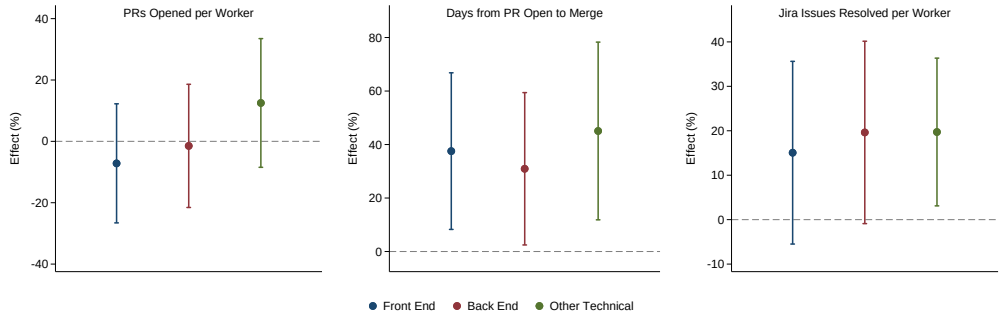
A. Baseline coding activity



B. Seniority



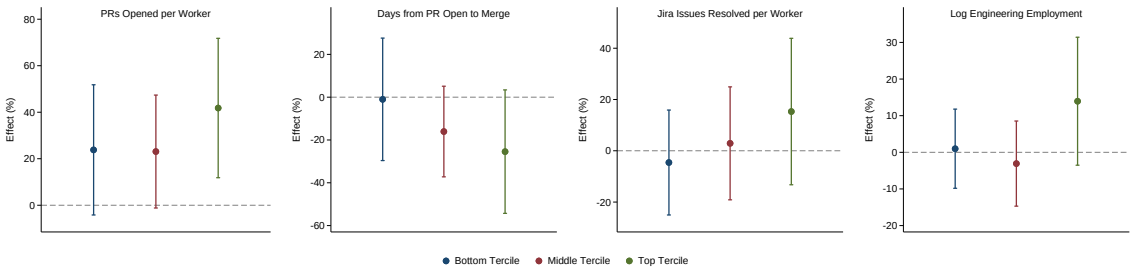
C. Software area



Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding agents, by worker characteristics. Each point is the event-study coefficient twelve months after adoption for one group of workers, as a percentage of that group's pre-adoption mean, with 95% confidence intervals clustered by firm. We construct a firm-by-worker group-by-month panel, assigning each worker to a single group for the whole sample period. PRs opened and Jira issues (tasks and stories) resolved per worker divide the group's total by the number of its workers active in JF that firm-month, winsorized at the 99th percentile. Days from PR open to merge is the mean over the group's merged pull requests. Each regression interacts the event-time indicators with the worker group, with firm-by-group and month fixed effects, baseline engineering employment interacted with month, and binned months since the firm's GitHub or Jira coverage began. The sample is the firms that adopt AI coding agents during our sample period. In Panel A, workers are split into terciles of pull requests opened in the baseline period (October 2023 to March 2024), among workers who opened at least one. In Panel B, software engineers are classified by job title as junior individual contributors, senior individual contributors, or engineering managers. In Panel C, workers are assigned the area, based on keywords in repository names, that accounts for most of the pull requests they opened before their firm adopted; other technical includes infrastructure, data, and mobile, and workers with non-technical job titles are excluded.

FIGURE A23. Heterogeneity by take-up intensity - AI assistants

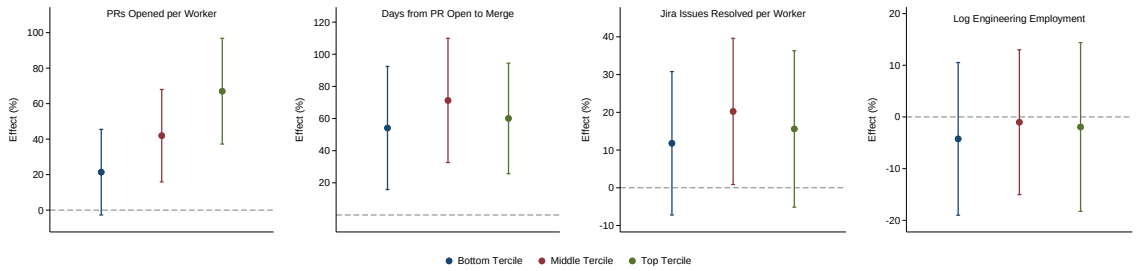
A. Share of engineers using AI assistants



Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding assistants, by how widely the firm's engineers take up the tool. Firms are split into terciles of the share of engineers committing code at the firm in the six months after adoption who personally adopt AI assistants within those six months. Because take-up is measured after adoption, these estimates describe a dose-response rather than a pre-determined characteristic. Each point is the event-study coefficient sixteen months after adoption for one group of firms, as a percentage of that group's pre-adoption mean (for log engineering employment, 100 times the coefficient), with 95% confidence intervals clustered by firm. PRs opened and Jira issues (tasks and stories) resolved per worker divide the firm's total by the number of its workers active in JF that month, winsorized at the 99th percentile; engineering employment is the number of workers active in JF. Days from PR open to merge is the mean over the firm's merged pull requests. Each regression interacts the event-time indicators with the firm group, with firm and month fixed effects, baseline engineering employment interacted with month, and binned months since the firm's GitHub or Jira coverage began. The sample is the firms that adopt AI coding assistants during our sample period.

FIGURE A24. Heterogeneity by take-up intensity - AI agents

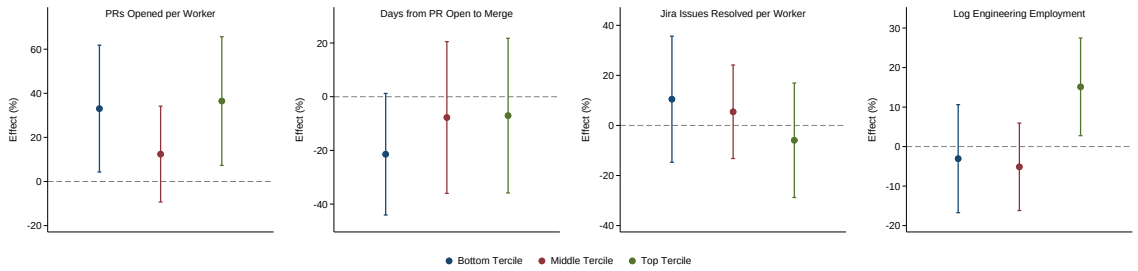
A. Share of engineers using AI agents



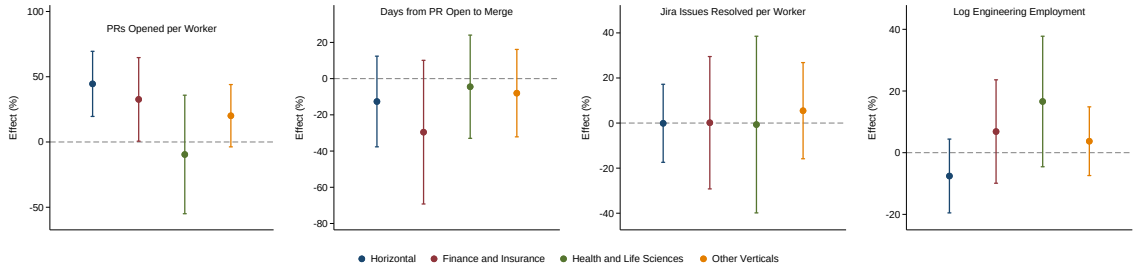
Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding agents, by how widely the firm's engineers take up the tool. Firms are split into terciles of the share of engineers committing code at the firm in the six months after adoption who personally adopt AI coding agents within those six months. Because take-up is measured after adoption, these estimates describe a dose-response rather than a pre-determined characteristic. Each point is the event-study coefficient twelve months after adoption for one group of firms, as a percentage of that group's pre-adoption mean (for log engineering employment, 100 times the coefficient), with 95% confidence intervals clustered by firm. PRs opened and Jira issues (tasks and stories) resolved per worker divide the firm's total by the number of its workers active in JF that month, winsorized at the 99th percentile; engineering employment is the number of workers active in JF. Days from PR open to merge is the mean over the firm's merged pull requests. Each regression interacts the event-time indicators with the firm group, with firm and month fixed effects, baseline engineering employment interacted with month, and binned months since the firm's GitHub or Jira coverage began. The sample is the firms that adopt AI coding agents during our sample period.

FIGURE A25. Heterogeneity by firm type - AI assistants

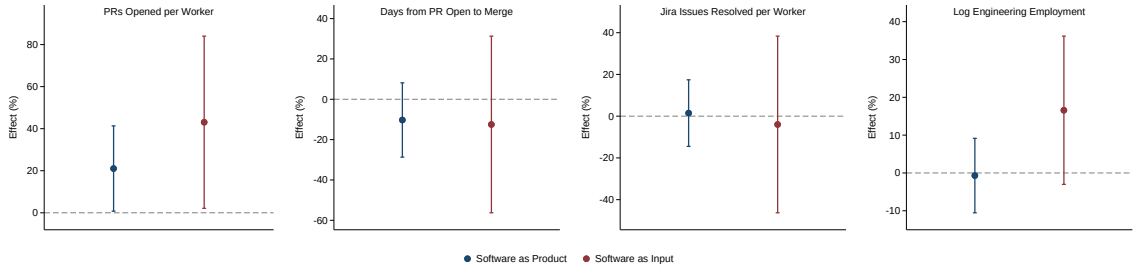
A. Firm size



B. Industry



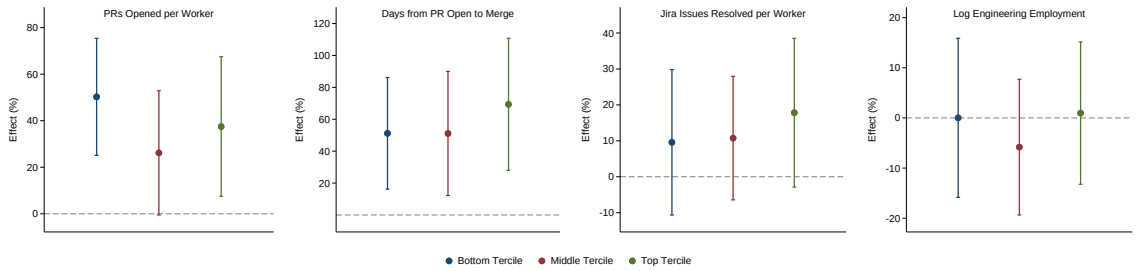
C. Software as input vs. as product



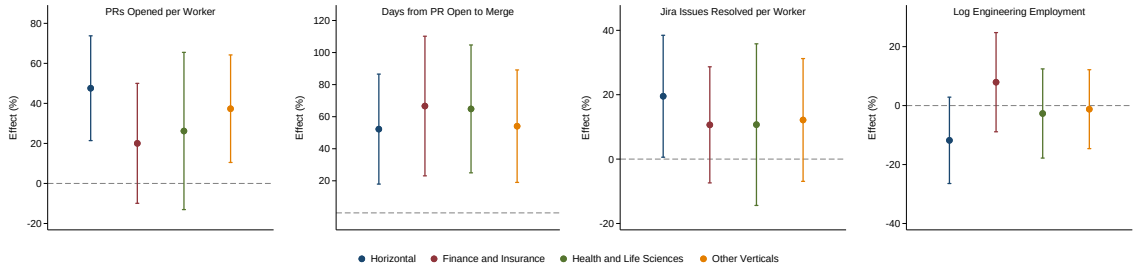
Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding assistants, by firm type. In Panel A, firms are split into terciles of total employment, the average number of LinkedIn profiles attached to the firm over the baseline period (January to June 2022). In Panel B, firms are grouped by the industry of their customers, classified from firm descriptions: horizontal software sold across industries, finance and insurance, health and life sciences, and other verticals, which includes firms without a coded industry. In Panel C, firms that sell software, as a license or as a digital service, are compared with firms that use software as an input to a non-software business; firms whose descriptions are too sparse to classify are excluded. Estimates, controls, and sample are as in Figure A23.

FIGURE A26. Heterogeneity by firm type - AI agents

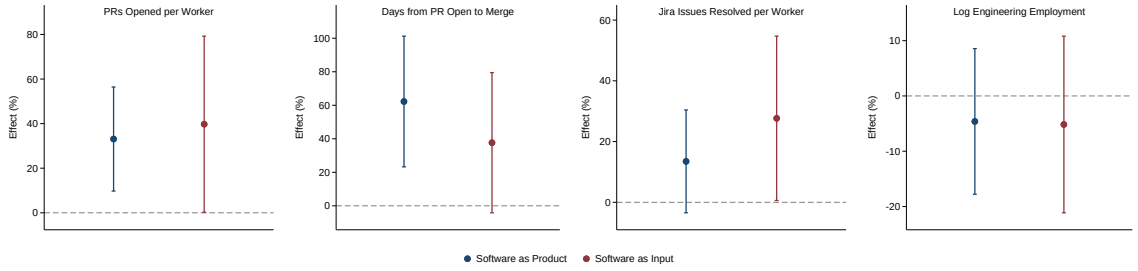
A. Firm size



B. Industry

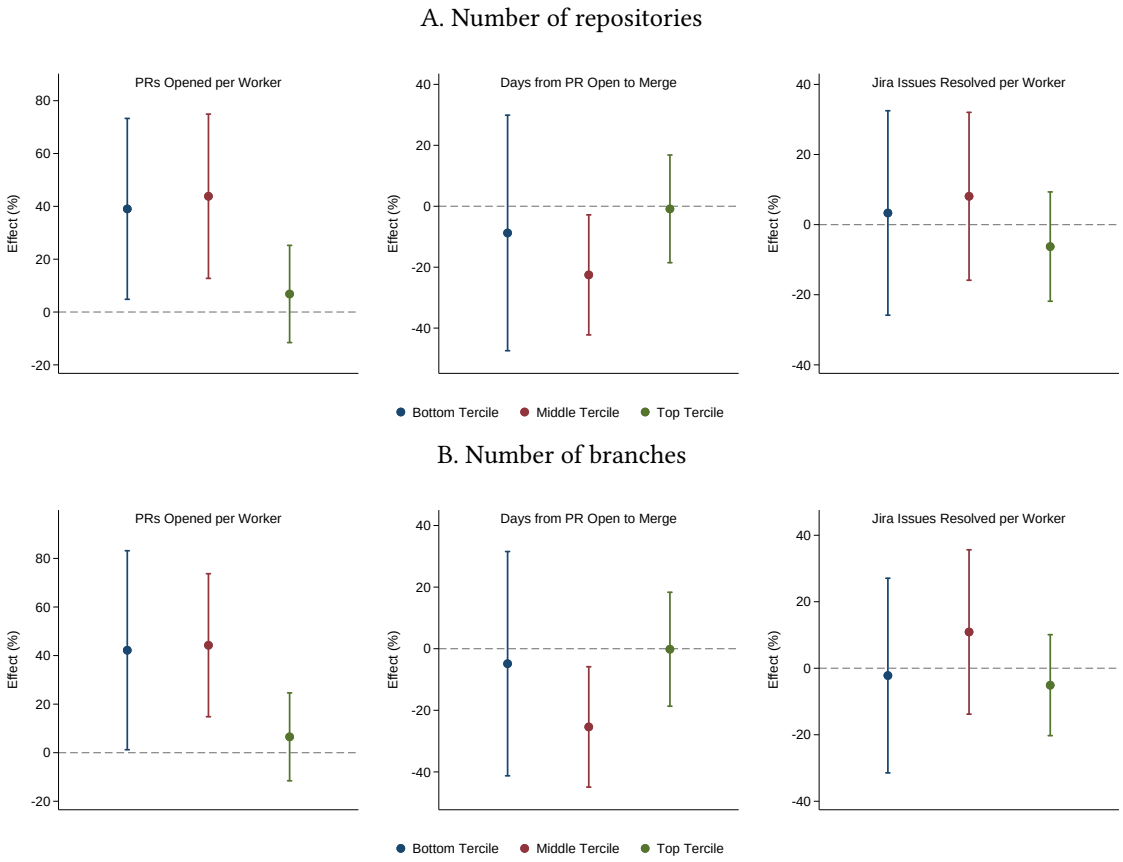


C. Software as input vs. as product



Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding agents, by firm type. In Panel A, firms are split into terciles of total employment, the average number of LinkedIn profiles attached to the firm over the baseline period (October 2023 to March 2024). In Panel B, firms are grouped by the industry of their customers, classified from firm descriptions: horizontal software sold across industries, finance and insurance, health and life sciences, and other verticals, which includes firms without a coded industry. In Panel C, firms that sell software, as a license or as a digital service, are compared with firms that use software as an input to a non-software business; firms whose descriptions are too sparse to classify are excluded. Estimates, controls, and sample are as in Figure A24.

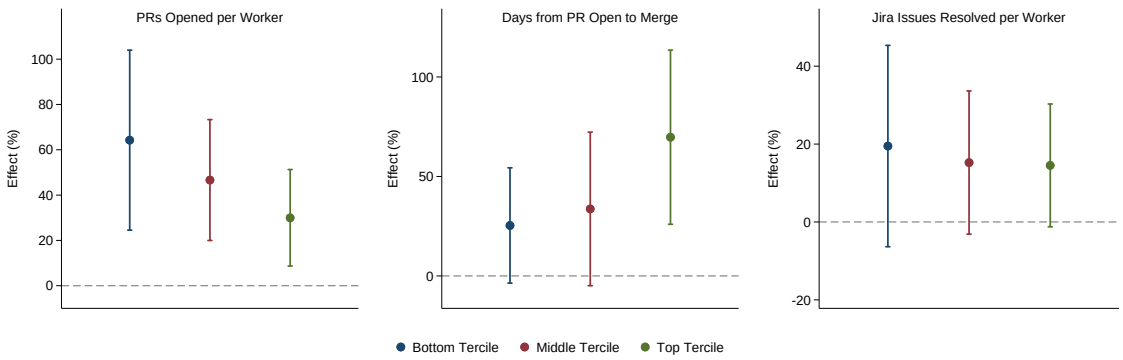
FIGURE A27. Heterogeneity by codebase complexity - AI assistants



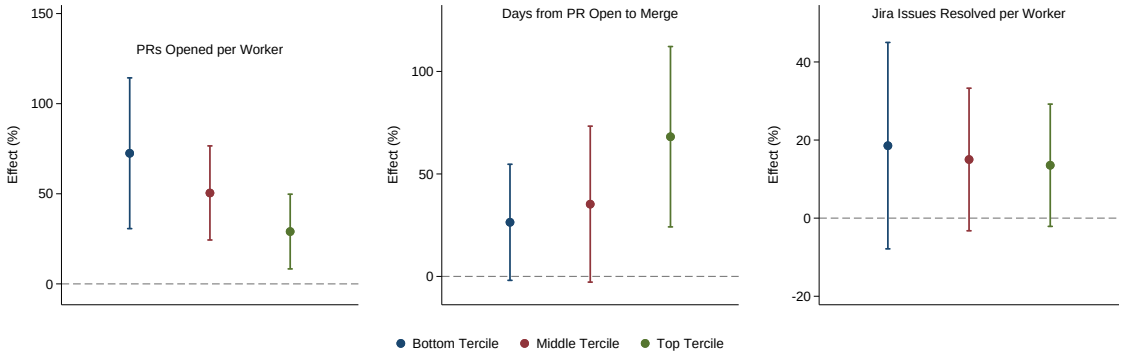
Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding assistants, by baseline codebase complexity. Firms are split into terciles of the number of distinct repositories (Panel A) or base branches (Panel B) that pull requests were opened against per month, averaged over the baseline period (January to June 2022). Estimates, controls, and sample are as in Figure A23.

FIGURE A28. Heterogeneity by codebase complexity - AI agents

A. Number of repositories



B. Number of branches



Notes: This figure displays heterogeneity in the effects of firm-level adoption of AI coding agents, by baseline codebase complexity. Firms are split into terciles of the number of distinct repositories (Panel A) or base branches (Panel B) that pull requests were opened against per month, averaged over the baseline period (October 2023 to March 2024). Estimates, controls, and sample are as in Figure A24.

E Appendix Tables

TABLE A1. Example Jira issues and labels

Task Description	Length
TASK-101: Implement OAuth 2.0 authentication system As a user, I want to log in using my Google or GitHub account so that I don't need to create a new password. Integrate OAuth 2.0 flow for Google and GitHub providers, create authentication middleware for token validation, implement session management and refresh token logic, add user account linking functionality for existing users, write unit and integration tests for auth flows, and update API endpoints to use new authentication system.	Long (8-12H)
TASK-102: Fix date format display in user profile The user's date of birth is displaying in MM/DD/YYYY format, but should display in DD/MM/YYYY format for European users. Update date formatting function to check user's locale settings and apply correct format in profile view component.	Short (<4H)
TASK-103: Conduct security audit and document remediation plan Perform a comprehensive security audit of the application to identify vulnerabilities and create a detailed remediation roadmap. Review all API endpoints for security vulnerabilities, analyze data storage and transmission practices, evaluate third-party dependencies for known CVEs, document all findings with severity ratings, create prioritized remediation plan with effort estimates, and present findings to engineering leadership.	Long (8-12H)
TASK-104: Update API documentation for new endpoints Three new endpoints were added in the last sprint but are not yet documented in our API docs. Document `/api/v2/notifications/preferences` (GET, PUT) and `/api/v2/notifications/history` (GET). Include request/response examples and parameter descriptions, and update Postman collection with new endpoints.	Short (<4H)

TABLE A2. Validation of supervised learning step

	MSE	MAE
Length	0.4105	0.5019

Notes: This table shows the validation of the supervised learning step for assigning the "length" label. Here, the true labels are the GPT-assigned labels, and the predicted labels are the labels assigned by the supervised learning model. We report the mean squared error and mean absolute error.

F Data Appendix

F.1 Measuring AI take-up

We construct firm- and individual-level AI tool adoption dates by combining two sources: (1) API data on business license seat activations collected by JF, and (2) signals derived from GitHub commit and pull request data.

Source 1: API data. JF's platform integrates directly with the APIs of three AI coding tools — GitHub Copilot, Cursor, and Claude Code. For each integration, the API provides the date at which the worker first uses the AI tool. We take the minimum of these dates as the first firm-level usage.

Tool codes in the raw data are mapped to two treatment groups as follows: *AI coding assistants* include GitHub Copilot (GC) and Cursor (CURS) (truncated to take-up measured pre-July 2024, to avoid conflation with take-up of AI coding agents); *AI coding agents* include Claude Code.

Source 2: GitHub signals. We extract three types of signals from the GitHub data.

Signal A — Bot account detection. We match GitHub user `login` and `name` fields against patterns for over 70 known AI tools using regular expressions. Each matched account is assigned to a tool name and category. The date of first appearance of a bot account of a given category within a firm provides the earliest signal of that tool type's use. The table below lists representative tools and the account identifiers used to match them.

AI bot accounts detected in GitHub data (Signal A)

Tool	**Login / name pattern**
Coding agents	
Cursor Agent	cursoragent
Devin	devin-ai-integration[bot]
GitHub Copilot Agent	copilot-swe-agent[bot]
Claude Code	claude-code[bot]
Factory AI	factory-droid[bot]
Open Hands	openhands-agent
Google Jules	google-labs-jules[bot]
Codegen	codegen-sh[bot]
Amp	amp-code
SWE-agent	swe-agent
Code review tools	
CodeRabbit	coderabbitai[bot]
Greptile	greptile-apps
Graphite	graphite-app
GitHub Copilot Reviewer	copilot-pull-request-reviewer
Cursor Bug Bot	cursor-com[bot]
Bito AI	bito-code-review
Qodo	qodo-merge
Sourcery	sourcery-ai
Korbit AI	korbit-ai

Signal B — Commit message signatures. We scan commit messages for text patterns associated with specific AI tools. This signal is classified as an agent signal, as it indicates that an AI tool has made direct code contributions.

Signal C — PR body signatures. We scan pull request body text for AI tool signatures. This signal is also classified as an agent signal. The table below lists the patterns used for both signals.

Commit message and PR body patterns used to detect AI tool usage (Signals B and C)

Tool	**Pattern**
Signal B: commit message patterns	
Claude Code	Co-authored-by: Claude + anthropic.com>
Claude Code	Generated with [Claude Code]
GitHub Copilot	Co-authored-by: Copilot
GitHub Copilot Agent	Co-authored-by: copilot-swe-agent
Cursor	Co-authored-by: Cursor
Cursor	cursorrules
Windsurf	Co-authored-by: Windsurf
Windsurf	windsurfrules / windsurf rules
Augment Code	Co-authored-by: Augment Code <support@augmentcode.com>
Gemini Code Assist	Co-authored-by: gemini-code-assist
Devin	app.devin.ai
OpenAI Codex	Generated by OpenAI Codex
Signal C: PR body patterns	
Claude Code	Generated with [Claude Code]
Augment Code	Pull request opened by [Augment Code]
Devin	Created by Devin / devin.ai
Factory AI	factory.ai / Created by Factory
OpenAI Codex	Generated by OpenAI Codex

F.2 Measuring Jira issue content

In this section, we describe our method for classifying Jira tasks based on their text descriptions. Each Jira task contains a "summary", a "description", a user-inputted "issue type", linked "project name", and linked "parent task". This task information can be manually entered by a worker (the "task creator") or automatically generated (e.g., an automatic bug report). On average, the text descriptions are 580 characters long, amounting to roughly one paragraph.

LLM labels. We begin by using the OpenAI API to classify a 0.5% random sample of tasks, stratified by company, on the estimated time an experienced software engineer would require to complete the task (*length*). We use the GPT-4o model with the following prompt:

```
You are a Jira task classification AI.
For each provided software engineering task, classify it according to the "length"
label defined below. Assume the perspective of a software engineer completing the
task, inferring implicit steps and context.

Length: The number of hours that an experienced software engineer
(i.e., 5+ years of work experience) would require to complete the task,
without the assistance of generative AI tools.
- 0 = Short: < 4 hours. Low complexity, routine, single-component change.
- 1 = Medium: >= 4 and < 8 hours. Medium complexity, multi-component change.
- 2 = Long: >= 8 hours and < 12 hours. High complexity, cross-team change.
- 3 = Very Long: >= 12 hours. Very high complexity, large scope change.

As output, provide the label in the following JSON format:
{"length": <int>}

Below is information about the task, including context about its parent task and project.
Classify the task itself, but use the context to inform your classification.
Project name: {project_name}
Parent task summary: {parent_task}
Task type: {issue_type}
Task summary: {summary}
Task description: {description}
```

Supervised extrapolation. We use a supervised learning approach to scale the GPT-assigned labels to the full dataset. First, we convert the Jira issue text descriptions into numerical embeddings using the Sentence-BERT (SBERT) model `all-MiniLM-L6-v2`, a transformer-based architecture designed for sentence-level encoding. Specifically, the model maps each issue's text description (title and summary) into a 384-dimensional numerical vector. Second, we train a neural network to predict the GPT-assigned length label from the embeddings, using PyTorch. We perform this training on the pre-AI adoption set of Jira issues, and then run the trained model on the entire set of Jira issues.

G Proofs Appendix

G.1 Proofs of main results

Proof of Proposition 1. Output is $Y = \frac{1}{n}\min\{C, R\}$, subject to $C \le C^P = A_C L_C$ and $R \le R^P = A_R L_R$, with $L_C + L_R = 1$ and $L_R \le \bar{L}_R$.

Step 1: all code and reviews produced are put into production. Y is weakly increasing in both C and R, which are bounded above only by C^P and R^P. Furthermore, with time discounting, delaying a unit lowers its present value. Thus, firms put all code and reviews into production, $C = C^P$ and $R = R^P$.

Step 2: the two stages are equalized. Substituting $L_R = 1 - L_C$, coding capacity $C^P = A_C L_C$ is increasing in L_C and review capacity $R^P = A_R[1 - L_C]$ is decreasing in it, so their minimum is maximized where they cross. To see formally that we cannot have an imbalance between the two stages, suppose first that $C^P > R^P$. Then $Y = R^P/n$, and lowering L_C by a small $\epsilon > 0$ raises R^P by $A_R\epsilon$ while lowering C^P by $A_C\epsilon$; for ϵ small enough the inequality still holds, so output rises. Thus, the firm cannot be at its optimum at an inequality. The equivalent will be true for $C^P < R^P$.

Step 3: the allocation and output. Setting $A_C L_C^* = A_R[1 - L_C^*]$ gives $L_C^*[A_C + A_R] = A_R$, so

$$L_C^* = \frac{A_R}{A_C + A_R} = \frac{1/A_C}{1/A_C + 1/A_R}, \qquad L_R^* = \frac{A_C}{A_C + A_R} = \frac{1/A_R}{1/A_C + 1/A_R},$$

which is item 1, and the two forms agree after multiplying numerator and denominator by $A_C A_R$. Substituting back,

$$C^{P*} = A_C L_C^* = \frac{A_C A_R}{A_C + A_R} = \frac{1}{1/A_C + 1/A_R},$$

and by Steps 1 and 2 this equals C^*, R^* and R^{P*}, with $Y^* = C^*/n$. This is item 2. Feasibility requires $L_R^* \le \bar{L}_R$, which is the premise.

Step 4: no backlog. Since $C_\tau^P = R_\tau^P$ at every τ, the backlog $B_t = \int_0^t [C_\tau^P - R_\tau^P]\,\mathrm{d}\tau = 0$, and hence $W_t = B_t/R_t^P = 0$. □

Proof of Lemma 1. *Step 1: the elasticities of capacity and time in review.* First, we take the elasticity of coding capacity with respect to A_C. Coding capacity is $C^P = A_C L_C$ with L_C held fixed, so $\ln C^P = \ln A_C + \ln L_C$ and $\partial \ln C^P / \partial \ln A_C = 1$. Next, review capacity is $R^P = A_R L_R$ and time in review is $1/A_R = 1/\left[[1-\pi]\tilde{A}_R\right]$; neither contains A_C, so both elasticities with respect to A_C are zero.

Step 2: the backlog and wait time. By Proposition 1, the firm begins with coding and review capacity balanced $C^P = R^P$, so raising A_C with the labor allocation fixed gives $C^P > R^P$. The backlog then accumulates at $\dot{B} = C^P - R^P > 0$. Since A_C does not affect R^P, differentiating $W_t = B_t/R^P$ gives

$\dot{W} = \dot{B}/R^P$.

Step 3: the interval to approval. Putting these together, the interval to approval is $W_t + 1/A_R$ and its second term is unchanged, so the whole of its increase comes from the first. □

Proof of Lemma 2. *Step 1: the elasticity of coding capacity.* Coding capacity $C^P = A_C L_C$ does not depend on π, so its elasticity with respect to π is 0.

Step 2: the elasticity of review productivity. Next, we take the elasticity of review productivity with respect to the bug rate. Taking logs of $A_R = [1-\pi]\tilde{A}_R$, we get $\ln A_R = \ln[1-\pi] + \ln \tilde{A}_R$. Since $\partial \ln \pi = \partial \pi/\pi$ and $\partial \ln[1-\pi]/\partial\pi = -1/[1-\pi]$,

$$\frac{\partial \ln A_R}{\partial \ln \pi} = \pi \cdot \frac{\partial \ln[1-\pi]}{\partial \pi} = -\frac{\pi}{1-\pi} \; < \; 0.$$

Review capacity is $R^P = A_R L_R$ with L_R fixed; thus, its elasticity is the same. Time in review is $1/A_R = A_R^{-1}$, so its elasticity is the negative of it, $+\pi/[1-\pi] > 0$.

Step 3: the backlog and wait time. By Proposition 1, the firm begins with coding and review capacity balanced $C^P = R^P$, so a fall in R^P with C^P unchanged gives $C^P > R^P$, and the backlog accumulates at $\dot{B} = C^P - R^P > 0$. AI adoption is a one-time shock, so R^P is still constant with respect to time post-adoption, and $\dot{W} = \dot{B}/R^P$ as before.

Step 4: the interval to approval. Both terms of $W_t + 1/A_R$ now rise: the second directly, by $\pi/[1-\pi]$, and the first indirectly because of the resulting difference in coding and review capacities. □

Proof of Proposition 2. *Item 1.* By Lemmas 1 and 2, either an increase in A_C or π leads coding capacity to outpace review capacity $C^P > R^P$ if the labor allocation is held fixed. Thus, output is constrained by review capacity $Y = \frac{1}{n}\min\{C^P, R^P\} = R^P/n$, the backlog accumulates at $\dot{B} = C^P - R^P > 0$, and $W_t = B_t/R^P$, which is item 1.

Item 2. Take $Y = R^P/n = A_R L_R/n$ with L_R fixed, so that $\ln Y = \ln A_R + \ln L_R - \ln n$. First, we take the elasticity of output with respect to A_C. Output does not depend on A_C, so $\partial \ln Y/\partial \ln A_C = 0$: the coding gain accumulates entirely in the backlog rather than reaching output. Next, the bug rate enters output only through A_R, so its elasticity with respect to the bug rate is that of A_R, which is $-\pi/[1-\pi]$ from Lemma 2. □

Proof of Proposition 3. *Step 1: the new optimum (item 3).* Define $L_R^{*\prime}$ as the firm's unconstrained optimal labor allocation post-AI adoption. This allocation is feasible when $L_R^{*\prime} \leq \bar{L}_R$. Applying Proposition 1, at that allocation, $C^P = R^P$, so $\dot{B}^* = 0$ and $\dot{W}^* = 0$.

Step 2: the elasticity of output with respect to each productivity. Output is $Y^* = \frac{1}{n}\left[1/A_C + 1/A_R\right]^{-1}$, so $\ln Y^* = -\ln n - \ln\left[1/A_C + 1/A_R\right]$. First, we take the elasticity of output with respect to A_C. The

derivative of $1/A_C$ with respect to $\ln A_C$ is $-1/A_C$, so

$$\frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln A_C} = \frac{1/A_C}{1/A_C + 1/A_R} = L_C^*, \qquad \text{and symmetrically} \qquad \frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln A_R} = \frac{1/A_R}{1/A_C + 1/A_R} = L_R^*,$$

where the last equality in each uses the shares from Proposition 1.

Step 3: the elasticity of output with respect to the bug rate (item 1). To get the elasticity with respect to π, we chain the second with the elasticity of A_R with respect to π through Lemma 2, which gives

$$\frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln \pi} = \frac{\mathrm{d}\ln Y^*}{\mathrm{d}\ln A_R} \cdot \frac{\mathrm{d}\ln A_R}{\mathrm{d}\ln \pi} = -L_R^* \frac{\pi}{1-\pi}.$$

Step 4: the relative labor allocation (item 2). From Proposition 1, $A_C L_C^* = A_R L_R^* \iff L_R^*/L_C^* = A_C/A_R$, so $\ln[L_R^*/L_C^*] = \ln A_C - \ln A_R$. Differentiating gives $\mathrm{d}\ln[L_R^*/L_C^*]/\mathrm{d}\ln A_C = 1$. Since the ratio falls one-for-one with A_R, using the elasticity of A_R with respect to π from Lemma 2, we get $\mathrm{d}\ln[L_R^*/L_C^*]/\mathrm{d}\ln \pi = \pi/[1-\pi]$. □

Proof of Proposition 4. *Step 1: the firm sets review labor at the ceiling (item 2).* If $L_R^{*\prime} > \bar{L}_R$ the equalizing allocation is infeasible. To see where the firm sets labor instead, substitute $L_C = 1 - L_R$, so that $Y = \frac{1}{n}\min\{A_C[1-L_R], A_R L_R\}$. For $L_R < L_R^{*\prime}$, review capacity is the smaller of the two, $A_R L_R < A_C[1-L_R]$, so $Y = A_R L_R/n$, which is increasing in L_R. Every feasible allocation has $L_R \le \bar{L}_R < L_R^{*\prime}$, so output is increasing in L_R over the whole feasible range, and the firm sets $L_R = \bar{L}_R$ and $L_C = 1 - \bar{L}_R$. Both are constants, not dependent on A_C or π, so the elasticities of both with respect to the two channels are 0.

Step 2: review is the binding stage. At that allocation $C^P = A_C[1-\bar{L}_R] > A_R\bar{L}_R = R^P$, where the inequality follows from $\bar{L}_R < L_R^{*\prime}$, so review is the binding stage and output is $Y^* = A_R\bar{L}_R/n$.

Step 3: the elasticities of output (item 1). Taking logs, $\ln Y^* = \ln A_R + \ln \bar{L}_R - \ln n$. First, we take the elasticity of output with respect to A_C. This does not depend on A_C, so $\mathrm{d}\ln Y^*/\mathrm{d}\ln A_C = 0$. Next, the bug rate enters only through A_R, so by Lemma 2 the bug rate still reaches output and does so undiluted, $\mathrm{d}\ln Y^*/\mathrm{d}\ln \pi = -\pi/[1-\pi]$: in the slack case it entered multiplied by L_R^*, here by one, because the review share no longer moves.

Step 4: the backlog (item 3). The backlog grows at $\dot{B}^* = A_C[1-\bar{L}_R] - A_R\bar{L}_R > 0$. Since the allocation is fixed at the ceiling, neither capacity changes over time, so this rate is constant and the backlog grows without bound, as does the wait time $W_t = B_t/R^P$. □

Proof of Lemma 3. The firm maximizes $PY - w[L_C + L_R]$, with $P = a - bY$ and $Y = \frac{1}{n}\min\{C, R\}$, subject to $C \le C^P = A_C L_C$ and $R \le R^P = A_R L_R$. As in Proposition 1, we take the review ceiling $\rho L_R \le \phi L$ to be slack at the optimum.

Step 1: all code and reviews produced are put into production. As in the proof of Proposition 1, profit is weakly increasing in C and R, and with time discounting, delaying a unit lowers its present value. Thus, $C = C^P$ and $R = R^P$, so $B_t = 0$ and $W_t = B_t / R_t^P = 0$, which is item 3.

Step 2: the two stages are equalized. Suppose $A_C L_C > A_R L_R$. Then $Y = A_R L_R / n$, and lowering L_C by a small $\epsilon > 0$ leaves output and the price unchanged while lowering the wage bill by $w\epsilon$, so profit rises. Thus, the firm cannot be at its optimum at an inequality, and the equivalent is true for $A_C L_C < A_R L_R$. At the optimum, $A_C L_C = A_R L_R = nY$, so

$$L_C = \frac{nY}{A_C}, \qquad L_R = \frac{nY}{A_R}, \qquad L = L_C + L_R = nY\Big[\frac{1}{A_C} + \frac{1}{A_R}\Big].$$

Dividing L_C and L_R by L gives the shares L_C^* and L_R^* of Proposition 1.

Step 3: profit as a function of output alone. Substituting the labor choices into the expression for profit, we get $[a - bY]Y - cY$. Additionally, the wage bill per unit of output is just the wage per unit of labor w times the amount of labor per unit of output L,

$$c = wn\Big[\frac{1}{A_C} + \frac{1}{A_R}\Big] = wn\,\frac{A_C + A_R}{A_C A_R}.$$

This does not depend on Y, because the technology has constant returns. Hence, $L = cY/w$.

Step 4: the choice of output. Differentiating profit with respect to Y and setting it to zero gives $a - 2bY = c$, so $Y = [a - c]/2b$. Substituting this into the demand function, we get $P = a - bY = [a + c]/2$. The second derivative is $-2b < 0$, so this is a maximum. Together with Step 2, this is item 2.

Step 5: the markup. Finally, we express the price P as a function of the wage bill c to get the markup. Taking the price elasticity of demand, $\varepsilon = P/[a - P] = [a + c]/[a - c]$, so $\varepsilon - 1 = 2c/[a - c]$ and $\varepsilon/[\varepsilon - 1] = [a + c]/2c$. Multiplying by c gives $P = \frac{\varepsilon}{\varepsilon - 1}\, c$, which is item 1. □

Proof of Proposition 5. By Lemma 3, both the coding productivity and bug rate channels affect output and labor only through the wage bill per unit of output c, so we first derive how c responds.

Step 1: the wage bill per unit of output. Taking logs of $c = wn[1/A_C + 1/A_R]$, with w and n fixed, and using that the derivative of $1/A_C$ with respect to $\ln A_C$ is $-1/A_C$,

$$\mathrm{d}\ln c = -\frac{1/A_C}{1/A_C + 1/A_R}\,\mathrm{d}\ln A_C - \frac{1/A_R}{1/A_C + 1/A_R}\,\mathrm{d}\ln A_R = -L_C^*\,\mathrm{d}\ln A_C - L_R^*\,\mathrm{d}\ln A_R.$$

Thus, $\mathrm{d}\ln c / \mathrm{d}\ln A_C = -L_C^*$. The bug rate enters only through A_R, whose elasticity with respect to π is $-\pi/[1 - \pi]$ by Lemma 2, so $\mathrm{d}\ln c / \mathrm{d}\ln \pi = L_R^*\,\pi/[1 - \pi]$.

Item 1. Taking logs of $Y = [a - c]/2b$ gives $\mathrm{d}\ln Y / \mathrm{d}\ln c = -c/[a - c] = -[\varepsilon - 1]/2$, using

$\varepsilon - 1 = 2c/[a - c]$ from the proof of Lemma 3. Multiplying by the two elasticities from Step 1,

$$\frac{\mathrm{d}\ln Y}{\mathrm{d}\ln A_C} = \frac{\varepsilon - 1}{2} L_C^* > 0, \qquad \frac{\mathrm{d}\ln Y}{\mathrm{d}\ln \pi} = -\frac{\varepsilon - 1}{2} L_R^* \frac{\pi}{1-\pi} < 0,$$

where the signs follow from $\varepsilon > 1$.

Item 2. Taking logs of $L = cY/w$ gives $\mathrm{d}\ln L = \mathrm{d}\ln c + \mathrm{d}\ln Y = \left[1 - \frac{\varepsilon-1}{2}\right]\mathrm{d}\ln c$. The first term is the displacement effect, as each unit of output requires less labor, and the second is the scale effect, as the firm produces more units. Substituting the elasticities of c from Step 1 gives item 2. Its sign turns on whether $[\varepsilon - 1]/2$ exceeds one, that is, on whether $\varepsilon > 3$.

Item 3. Taking logs of $L_C = nY/A_C$ and $L_R = nY/A_R$ gives

$$\mathrm{d}\ln L_C = \mathrm{d}\ln Y - \mathrm{d}\ln A_C, \qquad \mathrm{d}\ln L_R = \mathrm{d}\ln Y - \mathrm{d}\ln A_R.$$

For coding productivity, A_R does not move, so review labor responds only through output, $\mathrm{d}\ln L_R/\mathrm{d}\ln A_C = \frac{\varepsilon-1}{2} L_C^* > 0$, while coding labor also falls one-for-one with the labor needed per unit of code, $\mathrm{d}\ln L_C/\mathrm{d}\ln A_C = \frac{\varepsilon-1}{2} L_C^* - 1$. For the bug rate, A_C does not move, so coding labor responds only through output, $\mathrm{d}\ln L_C/\mathrm{d}\ln \pi = -\frac{\varepsilon-1}{2} L_R^* \frac{\pi}{1-\pi} < 0$. Review labor responds twice: it falls with output, but rises because A_R falls by $\pi/[1-\pi]$, so each unit of output requires more review labor. Thus, $\mathrm{d}\ln L_R/\mathrm{d}\ln \pi = \left[1 - \frac{\varepsilon-1}{2} L_R^*\right] \frac{\pi}{1-\pi}$. □

G.2 Model estimation

Moments. We convert the firm's problem in Section 3 to the six moments and two constraints in Section 6.

First, the technology is multiplicative, so we can get the first four moments from taking logs of the different elements of the set-up. This gives (M1) from $C^P = A_C L_C$, (M2) from $R^P = A_R L_R$, (M4) from $Y = R^P/n$, and (M3) from $A_R = [1-\pi]\tilde{A}_R$ with the raw review productivity $\tilde{A}_R$ held fixed.

Second, the remaining expressions compare or add up levels rather than logs, so we cannot use the same log transformation to simplify the change post-AI adoption. Thus, we write each post-adoption level relative to its pre-adoption optimum, times a multiple. For a variable that is strictly positive at both dates, let $\mathrm{d}\ln X \equiv \ln X' - \ln X^*$ denote its log change. Exponentiating both sides recovers the level,

$$\frac{X'}{X^*} = e^{\mathrm{d}\ln X} \qquad \Longleftrightarrow \qquad X' = X^* e^{\mathrm{d}\ln X},$$

so $e^{\mathrm{d}\ln X}$ is the post-adoption value as a multiple of its baseline and $e^{\mathrm{d}\ln X} - 1$ is the proportional change. Thus, taking the definitions of the queue moments, together with the fact that the firm starts out balanced at $C^{P*} = R^{P*}$, we get:

$$\dot{B} = C^{P\prime} - R^{P\prime} = C^{P*} e^{\mathrm{d}\ln C^P} - R^{P*} e^{\mathrm{d}\ln R^P} = C^{P*}\left[e^{\mathrm{d}\ln C^P} - e^{\mathrm{d}\ln R^P}\right],$$
$$\dot{W} = \frac{\dot{B}}{R^{P\prime}} = \frac{C^{P*}\left[e^{\mathrm{d}\ln C^P} - e^{\mathrm{d}\ln R^P}\right]}{R^{P*} e^{\mathrm{d}\ln R^P}} = e^{\mathrm{d}\ln C^P - \mathrm{d}\ln R^P} - 1,$$

which are (M5) and (M6). Constraint (C1) is the same step applied to the labor shares, which must still sum to one after adoption, so $L_R^*[e^{\mathrm{d}\ln L_R} - 1] = -L_C^*[e^{\mathrm{d}\ln L_C} - 1]$.

Measured moments. We measure the six elements of $\hat{m}$ from our estimates of the effects of adoption of AI agents in Section 4. For the percent changes, we estimate them from the last post-period of the event study. For the rates, we estimate them from the slope of the event study (i.e., based on the last and first post-periods of the event studies). Finally, for the change in bug rate, we simply take the change in levels.

Moment	Measured from	Value
$\mathrm{d}\ln C^P$	Pull requests created per worker-month	0.306
$\mathrm{d}\ln R^P$	Pull requests merged per worker-month	0.231
$\mathrm{d}\ln Y$	Tasks and stories resolved per worker-month	0.129
$\dot{B}$	Open unreviewed requests per worker-month	0.043
$\dot{W}$	Days from opening to first human comment	0.005
$\pi \to \pi'$	Share of requests drawing a comment	$0.352 \to 0.398$

Calibrated inputs. For the five elements of ψ, we measure them descriptively in our data.

Input	Source	Value
L_C^*	$1 - L_R^*$	0.657
L_R^*	Mean of three measures of review's share of coding-plus-review time, from the Jellyfish imputation of time allocation, Prolific survey of engineers, and Kumar et al. (2024).	0.343
C^{P*}	Pull requests merged per worker-month, pre-period	4.16
R^{P*}	$= C^{P*}$ at the pre-adoption optimum, by Proposition 1	4.16
n^*	C^{P*}/Y^*	1.63

Identification. We recover four free parameters θ from $\hat{m}$, ψ. The identification of each works as follows:

$$
\begin{aligned}
&\text{(M3)} && \mathrm{d}\ln A_R = \ln[1-\pi'] - \ln[1-\pi], \\
&\text{(M2)} && \mathrm{d}\ln L_R = \mathrm{d}\ln R^P - \mathrm{d}\ln A_R, \\
&\text{(C1)} && \mathrm{d}\ln L_C = \ln\left[1 - \frac{L_R^*}{L_C^*}\left[e^{\mathrm{d}\ln L_R} - 1\right]\right], \\
&\text{(M1)} && \mathrm{d}\ln A_C = \mathrm{d}\ln C^P - \mathrm{d}\ln L_C, \\
&\text{(M4)} && \mathrm{d}\ln n \ = \mathrm{d}\ln R^P - \mathrm{d}\ln Y.
\end{aligned}
$$

The constraint (C1) gives $\mathrm{d}\ln L_R$ from $\mathrm{d}\ln L_C$. (M5) and (M6) are the two over-identifying restrictions. Both reduce to $\mathrm{d}\ln C^P - \mathrm{d}\ln R^P$ to first order. They jointly impose:

$$
\frac{\dot{B}}{C^{P*}} \ = \ \dot{W},
$$

In practice, the estimator chooses the set of parameters $\hat{\theta}$ that minimizes the objective over all six moments at once.

G.3 Future impacts

This section describes the details of each simulation exercise in Section 6.3.

G.3.1 Long-run responses to the demand elasticity

Proposition 5 gives the long-run responses of output and labor to a marginal change in coding productivity or the bug rate. The changes from AI adoption are large, so here we derive the exact responses to a discrete change, which we plot in Figure 9. As in Section G.2, let X' denote the post-adoption value of a variable X and $\mathrm{d}\ln X \equiv \ln X' - \ln X$ its log change, so that $X'/X = e^{\mathrm{d}\ln X}$. By Lemma 2, the bug rate affects the firm only through A_R, so we state everything in terms of A_C and A_R rather than A_C and π. Throughout, we hold n at its pre-adoption level.

Step 1. The change in labor per unit of output. First, we calculate the change in the labor the firm needs per unit of output. By item 1 of Lemma 3, this is $c/w = n\,[1/A_C + 1/A_R]$, where c is the wage bill per unit of output. Since w and n are unchanged, c changes in the same proportion. Taking the ratio of its post- and pre-adoption values, and using the labor shares $L_C^* = [1/A_C]/[1/A_C + 1/A_R]$ and $L_R^* = [1/A_R]/[1/A_C + 1/A_R]$ from Proposition 1,

$$e^{\mathrm{d}\ln c} = \frac{1/A_C' + 1/A_R'}{1/A_C + 1/A_R} = L_C^*\, e^{-\mathrm{d}\ln A_C} + L_R^*\, e^{-\mathrm{d}\ln A_R}.$$

That is, labor per unit of output falls with each productivity gain, weighted by the share of labor it applies to. To first order, $\mathrm{d}\ln c \approx -L_C^*\,\mathrm{d}\ln A_C - L_R^*\,\mathrm{d}\ln A_R$, which is the expression in Step 1 of the proof of Proposition 5. For the large changes from adoption, we use the exact expression above.

Step 2. The demand curve in terms of the elasticity. Next, we write the demand intercept a in terms of the elasticity ε, which is the object we would like to vary. By item 2 of Lemma 3, $Y = [a - c]/2b$, and the corresponding price is $P = [a + c]/2$. The elasticity of linear demand at this point is then

$$\varepsilon = \frac{P}{a - P} = \frac{a + c}{a - c} \qquad \Longrightarrow \qquad a = c\,\frac{\varepsilon + 1}{\varepsilon - 1}.$$

Since $a > c$ whenever output is positive, $\varepsilon > 1$ at any interior optimum.

Step 3. Output. Output is linear in c, so we can take the ratio of its post- and pre-adoption levels

directly. The slope b enters both levels identically and cancels, leaving

$$
\begin{aligned}
e^{\mathrm{d}\ln Y} &= \frac{[a-c']/2b}{[a-c]/2b} = \frac{a-c'}{a-c} && \text{[Lemma 3, before and after adoption]} \\
&= \frac{\frac{\varepsilon+1}{\varepsilon-1} - e^{\mathrm{d}\ln c}}{\frac{\varepsilon+1}{\varepsilon-1} - 1} && \text{[substituting } a \text{ from Step 2; the common factor } c \text{ cancels]} \\
&= 1 + \frac{1}{2}\left[1 - e^{\mathrm{d}\ln c}\right][\varepsilon - 1] && \text{[multiplying through by } \varepsilon - 1 \text{ and rearranging]}.
\end{aligned}
$$

Step 4. Labor. By item 2 of Lemma 3, each labor margin is output times the labor the firm needs per unit of output: $L = cY/w$, $L_C = nY/A_C$, and $L_R = nY/A_R$. Since w and n are unchanged, taking log changes gives

$$
\mathrm{d}\ln L = \mathrm{d}\ln c + \mathrm{d}\ln Y, \qquad \mathrm{d}\ln L_C = -\mathrm{d}\ln A_C + \mathrm{d}\ln Y, \qquad \mathrm{d}\ln L_R = -\mathrm{d}\ln A_R + \mathrm{d}\ln Y.
$$

Thus, each labor margin changes with output and with the labor it needs per unit of output.

Step 5. The thresholds. Finally, we calculate the elasticity above which each labor margin rises. Each margin is unchanged when the expansion in output exactly offsets its fall in labor per unit of output. Setting each expression in Step 4 equal to zero and substituting $e^{\mathrm{d}\ln Y}$ from Step 3, overall labor rises when

$$
\varepsilon > 1 + 2\, e^{-\mathrm{d}\ln c},
$$

and coding labor rises when

$$
\varepsilon > 1 + 2\,\frac{e^{\mathrm{d}\ln A_C} - 1}{1 - e^{\mathrm{d}\ln c}}.
$$

At our estimates, $e^{\mathrm{d}\ln c} = 0.747$, so the two thresholds are $\varepsilon = 3.68$ and $\varepsilon = 6.83$. For review labor, the threshold is the same as for coding labor, with $\mathrm{d}\ln A_R$ in place of $\mathrm{d}\ln A_C$. Since $\mathrm{d}\ln A_R < 0$ at our estimates, this threshold is below one, so review labor rises for any $\varepsilon > 1$.

Step 6. The marginal case. As the changes become small, $1 - e^{\mathrm{d}\ln c} \approx L_C^*\,\mathrm{d}\ln A_C + L_R^*\,\mathrm{d}\ln A_R$. Holding A_R fixed, the two thresholds approach $\varepsilon = 3$ and $\varepsilon = 1 + 2/L_C^*$, which are the conditions in Proposition 5. Both thresholds we report in Section 6 are above these marginal values. □

G.3.2 Trajectory of coding productivity

This section describes the construction of Figure 10, Panel A.

Step 1. Technical capabilities. First, we take the 50% task-completion time horizons that Kwa et al. (2025) report for frontier AI models (Time Horizon 1.1). We keep the frontier models: those released

from January 2023 onward whose time horizon exceeds that of every earlier model. Following METR, we drop measurements above 16 hours, which METR considers unreliable with its current task suite. Next, we regress the log of the frontier of time horizon that can be completed by a model H on the model's release date t, measured in years,

$$\ln H_t = \alpha + \beta\, t,$$

so that the time horizon doubles every $\ln 2/\beta$ years, or every 129 days. This matches the doubling time METR reports for models released since 2023. The grey line plots the fitted time horizon relative to its level in February 2023, and the dots plot the measurements on the same scale.

Step 2. The two estimates of coding productivity. In order to estimate a trajectory through A_C, we need to place these at specific date points. We place our two estimates of A_C at the median adoption months of the two tools: April 2024 for AI assistants, T_A, and May 2025 for AI agents, T_G. For AI assistants, we apply the identification steps of Section 6 to the assistant-arm event study estimates at the end of their window — pull requests created, pull requests merged, and the share of pull requests that receive any comment. This gives the log change in coding productivity from AI assistants, $\mathrm{d}\ln A_C^{\text{assist}}$, with $e^{\mathrm{d}\ln A_C^{\text{assist}}} = 1.35$. For AI agents, we take the estimated log change in Table 5, $\mathrm{d}\ln A_C^{\text{agent}}$, with $e^{\mathrm{d}\ln A_C^{\text{agent}}} = 1.74$. We stack the two estimates, since the agent arm's pre-period is one in which firms already use AI assistants: relative to the pre-AI period, A_C at the agent adoption date has risen by a factor of $e^{\mathrm{d}\ln A_C^{\text{assist}} + \mathrm{d}\ln A_C^{\text{agent}}} = 2.34$.

Finally, we fit an exponential trend through the two estimates, holding A_C at its pre-AI level before the trend reaches it. Writing $\mathrm{d}\ln A_{C,t}$ for the log change in coding productivity at date t, relative to the pre-AI period,

$$\mathrm{d}\ln A_{C,t} = \max\left\{\mathrm{d}\ln A_C^{\text{assist}} + \frac{t - T_A}{T_G - T_A}\,\mathrm{d}\ln A_C^{\text{agent}},\ 0\right\}.$$

Since $\ln H$ is linear in t, this is the same as assuming that $\ln A_C$ is linear in $\ln H$: each doubling of the time horizon raises A_C by 19.7%.

G.3.3 Trajectories of output and employment

This section describes the construction of Figure 10, Panels B and C. Both take the path of $A_{C,t}$ from Section G.3.2.

Step 1. Review productivity. First, we specify the path of A_R. In Panel B, A_R is held at its post-adoption estimate throughout. In Panel C, A_R is held at this level until the agent adoption date, and then grows at the same rate as A_C. To prevent a discrete jump in the figure, we let A_R increase smoothly over the following year.

Step 2. Output per worker with labor held fixed. Next, we calculate output per worker with overall labor held fixed. By Proposition 1, the firm balances coding and review capacity, so that $L_{R,t} = A_{C,t}/[A_{C,t} + A_{R,t}]$, subject to the review ceiling $L_R \leq \bar{L}_R = 0.823$. Output per worker is then

$$y_t = \frac{A_{R,t}}{n_t} \min\left\{\frac{A_{C,t}}{A_{C,t} + A_{R,t}},\ \bar{L}_R\right\}.$$

Once L_R reaches the review ceiling $\bar{L}_R$, further increases in A_C no longer raise output per worker, and the additional code goes into the backlog. In Panel C, the review share stays at its value at the agent adoption date, so the ceiling never binds.

Step 3. The number of code changes per unit of output. As in Section 6, n is the residual between code output and final output, and we let it absorb what the balanced-allocation path misses at our two estimates. However, in the longer run, we are only interested in effects that come through product demand and the review ceiling. Thus, we fit a curve that allows $\ln n$ to change smoothly, rising with $\ln A_C$ along a concave curve that starts at zero at the pre-AI level, and choose its curvature and its limit so that output per worker passes through our measured output effects at T_A and T_G. Between T_G and January 2027, $\ln n$ flattens smoothly to a constant, and from then on, n is held fixed. Thus, from 2027, the trajectories move only through product demand and the review ceiling.

Step 4. Overall labor and output. Through December 2026, overall labor is held at its pre-AI level, so output equals output per worker, consistent with our estimates of the short- to medium-run overall employment effects. From January 2027, we allow labor to adjust flexibly, given the results of Section G.3.1. Here, the change in labor per unit of output is equal to $1/y_t$ and ε at the mean of the software estimates, 2.27:

$$L_t^* = \frac{1}{y_t}\left[1 + \frac{1}{2}\left[1 - \frac{1}{y_t}\right][\varepsilon - 1]\right].$$

We model this labor adjustment process as occurring gradually, such that there is not a discrete jump in January 2027. Specifically, labor closes half of the remaining gap to this level each year, with the speed of adjustment rising smoothly from zero over 2027. Output is then output per worker times workers, $Y_t = y_t L_t$, which holds in both regimes, so neither series jumps when labor becomes flexible.

Step 5. Holding output at its peak in Panel B. With A_R fixed, the long-run level of output under linear demand is below the level the firm reaches with labor held fixed, so output would fall back as labor adjusts. In Panel B, we instead hold output at its running maximum, $Y_t = \max_{\tau \leq t} y_\tau L_\tau$, and set employment to the labor that this output requires, $L_t = Y_t/y_t$. Employment therefore keeps falling while output per worker rises, and is constant once the review ceiling binds. In Panel C, output does not fall back, so this step does not bind.

References

Acemoglu, D. (2024, May). *The Simple Macroeconomics of AI* [Working Paper]. National Bureau of Economic Research. Retrieved 2024-11-14, from https://www.nber.org/papers/w32487 doi: 10.3386/w32487

Acemoglu, D., & Restrepo, P. (2018, January). *Artificial Intelligence, Automation and Work* [Working Paper]. National Bureau of Economic Research. Retrieved 2022-12-13, from https://www.nber.org/papers/w24196 doi: 10.3386/w24196

Acemoglu, D., & Restrepo, P. (2019, May). Automation and New Tasks: How Technology Displaces and Reinstates Labor. *Journal of Economic Perspectives*, *33*(2), 3–30. Retrieved 2022-10-05, from https://www.aeaweb.org/articles?id=10.1257/jep.33.2.3 doi: 10.1257/jep.33.2.3

Aghion, P., Jones, B. F., & Jones, C. I. (2017, October). *Artificial Intelligence and Economic Growth.*

Agrawal, A., Gans, J. S., & Goldfarb, A. (2024). Artificial intelligence adoption and system-wide change. *Journal of Economics & Management Strategy*, *33*(2), 327–337. Retrieved 2026-06-30, from https://onlinelibrary.wiley.com/doi/abs/10.1111/jems.12521 (_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jems.12521) doi: 10.1111/jems.12521

Anthony, C. (2021). When knowledge work and analytical technologies collide: The practices and consequences of black boxing algorithmic technologies. *Administrative Science Quarterly*, *66*(4), 1173–1212. doi: 10.1177/00018392211016755

Anthony, C., Bechky, B. A., & Fayard, A.-L. (2023). "collaborating" with AI: Taking a system view to explore the future of work. *Organization Science*, *34*(5), 1672–1694. doi: 10.1287/orsc.2022.1651

Aral, S., Brynjolfsson, E., & Wu, L. (2012, March). Three-Way Complementarities: Performance Pay, Human Resource Analytics, and Information Technology. *Management Science*, *58*(5). Retrieved 2026-06-30, from https://pubsonline.informs.org/doi/abs/10.1287/mnsc.1110.1460

Atlassian. (2026). *Learn the essentials of software development.* Retrieved 2026-07-14, from https://www.atlassian.com/pl/software-development

Autor, D. (2024, February). *Applying AI to Rebuild Middle Class Jobs* [Working Paper]. National Bureau of Economic Research. Retrieved 2024-11-14, from https://www.nber.org/papers/w32140 doi: 10.3386/w32140

Babina, T., Fedyk, A., He, A., & Hodson, J. (2024, January). Artificial intelligence, firm growth, and product innovation. *Journal of Financial Economics*, *151*, 103745. Retrieved 2024-10-09, from https://www.sciencedirect.com/science/article/pii/S0304405X2300185X doi: 10.1016/j.jfineco.2023.103745

Baldwin, C. Y., & Clark, K. B. (2000). *Design rules, volume 1: The power of modularity.* Cambridge, MA: MIT Press. doi: 10.7551/mitpress/2366.001.0001

Beane, M., & Anthony, C. (2024). Inverted apprenticeship: How senior occupational members develop practical expertise and preserve their position when new technologies arrive. *Organization Science*, *35*(2), 405–431. doi: 10.1287/orsc.2023.1688

Becker, J., Rush, N., Barnes, E., & Rein, D. (2025, July). *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity.* arXiv. Retrieved 2025-12-15, from http://arxiv.org/abs/2507.09089 (arXiv:2507.09089 [cs]) doi: 10.48550/arXiv.2507.09089

Berg, J., Raj, M., & Seamans, R. (2023). Capturing Value from Artificial Intelligence. *Academy of Management Discoveries*, *9*.

Besiroglu, T., Emery-Xu, N., & Thompson, N. (2024). Economic impacts of AI-augmented R&D. *Research Policy*, *53*(7), 105037. doi: 10.1016/j.respol.2024.105037

Borusyak, K., Jaravel, X., & Spiess, J. (2024, November). Revisiting Event-Study Designs: Robust and Efficient Estimation. *The Review of Economic Studies*, *91*(6), 3253–3285. Retrieved 2024-11-14, from https://doi.org/10.1093/restud/rdae007 doi: 10.1093/restud/rdae007

Bresnahan, T., Brynjolfsson, E., & Hitt, L. M. (2002, February). Information Technology, Workplace Organization, and the Demand for Skilled Labor: Firm-Level Evidence*. *The Quarterly Journal of Economics*, *117*(1), 339–376. Retrieved 2024-10-10, from https://doi.org/10.1162/003355302753399526 doi: 10.1162/003355302753399526

Briggs, J., & Kodnani, D. (2023, March). *The potentially large effects of artificial intelligence on economic growth* (Global Economics Analyst). Goldman Sachs.

Brynjolfsson, E., Chandar, B., & Chen, R. (2025). Canaries in the coal mine? Six facts about the recent employment effects of artificial intelligence. *Stanford Digital Economy Lab. Published August*.

Brynjolfsson, E., Li, D., & Raymond, L. (2025, May). Generative AI at Work. *The Quarterly Journal of Economics*, *140*(2), 889–942. Retrieved 2025-07-09, from https://doi.org/10.1093/qje/qjae044 doi: 10.1093/qje/qjae044

Brynjolfsson, E., Rock, D., & Syverson, C. (2018, October). *The Productivity J-Curve: How Intangibles Complement General Purpose Technologies* [Working Paper]. National Bureau of Economic Research. Retrieved 2026-06-30, from https://www.nber.org/papers/w25148 doi: 10.3386/w25148

Callaway, B., & Sant'Anna, P. H. C. (2021). Difference-in-differences with multiple time periods. *Journal of Econometrics*, *225*(2), 200–230.

Catling, H. (1970). *The spinning mule.* Newton Abbot: David & Charles.

Chamberlain, G. (1984). Panel data. In Z. Griliches & M. D. Intriligator (Eds.), *Handbook of econometrics* (Vol. 2, pp. 1247–1318). Elsevier. doi: 10.1016/S1573-4412(84)02014-6

Chapman, S. D. (1972). *The cotton industry in the industrial revolution.* London: Macmillan.

Chatterji, A., Cunningham, T., Deming, D. J., Hitzig, Z., Ong, C., Shan, C. Y., & Wadman, K. (2025, September). How People Use ChatGPT.

Chatterji, A., Holtz, D., Rakholia, N., Tambe, P., & Weeratunga, G. (2026, August). *How Organizations Use AI: Evidence from ChatGPT.*

Chen, Z., & Chan, J. (2024, December). Large Language Model in Creative Work: The Role of Collaboration Modality and User Expertise. *Management Science*, *70*(12), 9101–9117. Retrieved 2025-05-06, from https://pubsonline.informs.org/doi/10.1287/mnsc.2023.03014 (Publisher: INFORMS) doi: 10.1287/mnsc.2023.03014

Choi, J. H., & Schwarcz, D. (2023, August). *AI Assistance in Legal Analysis: An Empirical Study* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-05-09, from https://papers.ssrn.com/abstract=4539836 doi: 10.2139/ssrn.4539836

Chui, M., Hazan, E., Roberts, R., Singla, A., Smaje, K., Sukharevsky, A., … Zemmel, R. (2023, June). *The economic potential of generative AI: The next productivity frontier* (Tech. Rep.). McKinsey & Company. https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/

the-economic-potential-of-generative-ai-the-next-productivity-frontier.
Cillo, P., & Rubera, G. (2025, May). Generative AI in innovation and marketing processes: A roadmap of research opportunities. *Journal of the Academy of Marketing Science*, *53*(3), 684–701. Retrieved 2026-07-16, from https://doi.org/10.1007/s11747-024-01044-7 doi: 10.1007/s11747-024-01044-7
Cui, K. Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2024, March). The Productivity Effects of Generative AI: Evidence from a Field Experiment with GitHub Copilot. *An MIT Exploration of Generative AI*. Retrieved 2024-07-23, from https://mit-genai.pubpub.org/pub/v5iixksv/release/2 (Publisher: MIT) doi: 10.21428/e4baedd9.3ad85f1c
Czarnitzki, D., Fernández, G. P., & Rammer, C. (2023). Artificial intelligence and firm-level productivity. *Journal of Economic Behavior & Organization*, *211*, 188–205. doi: 10.1016/j.jebo.2023.05.008
Davidson, T. (2021, June). *Could advanced AI drive explosive economic growth?* (Tech. Rep.). Open Philanthropy. https://coefficientgiving.org/research/could-advanced-ai-drive-explosive-economic-growth/.
Davidson, T., Halperin, B., Houlden, T., & Korinek, A. (2026, October). When Does Automating Research Produce Explosive Growth?
de Chaisemartin, C., & D'Haultfœuille, X. (2024, February). Difference-in-Differences Estimators of Intertemporal Treatment Effects. *The Review of Economics and Statistics*, *1*(45). Retrieved 2025-11-07, from https://direct.mit.edu/rest/article-abstract/doi/10.1162/rest_a_01414/119488/Difference-in-Differences-Estimators-of?redirectedFrom=fulltext
Dell'Acqua, F., Ayoubi, C., Lifshitz-Assaf, H., Sadun, R., Mollick, E. R., Mollick, L., … Lakhani, K. R. (2025, March). *The Cybernetic Teammate: A Field Experiment on Generative AI Reshaping Teamwork and Expertise* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-05-01, from https://papers.ssrn.com/abstract=5188231 doi: 10.2139/ssrn.5188231
Dell'Acqua, F., McFowland III, E., Mollick, E. R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., … Lakhani, K. R. (2023, September). *Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-05-01, from https://papers.ssrn.com/abstract=4573321 doi: 10.2139/ssrn.4573321
Demirer, M., Horton, J. J., Immorlica, N., Lucier, B., & Shahidi, P. (2026, February). *CHAINING TASKS, REDEFINING WORK: A THEORY OF AI AUTOMATION.*
Demirer, M., Musolff, L., & Yang, L. (2026, May). *Writing Code vs. Shipping Code: Productivity Effects Across Generations of AI Coding Tools* [Working Paper]. National Bureau of Economic Research. Retrieved 2026-06-19, from https://www.nber.org/papers/w35275 doi: 10.3386/w35275
Derdenger, T. (2014). Technological tying and the intensity of price competition: An empirical analysis of the video game industry. *Quantitative Marketing and Economics*, *12*(2), 127–165. doi: 10.1007/s11129-014-9143-9
de Souza, G. (2025, July). *Artificial Intelligence in the Office and the Factory: Evidence from Administrative Software Registry Data* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-10-21, from https://papers.ssrn.com/abstract=5375463 doi:

10.2139/ssrn.5375463

Dillon, E. W., Jaffe, S., Immorlica, N., & Stanton, C. T. (2025). *Shifting work patterns with generative ai* (Tech. Rep.). National Bureau of Economic Research.

Dixon, J., Hong, B., & Wu, L. (2021). The robot revolution: Managerial and employment consequences for firms. *Management Science*, *67*(9), 5586–5605. doi: 10.1287/mnsc.2020.3812

Eisfeldt, A. L., Schubert, G., & Zhang, M. B. (2023). *Generative AI and firm values* (Working Paper No. 31222). National Bureau of Economic Research. doi: 10.3386/w31222

Eloundou, T., Manning, S., Mishkin, P., & Rock, D. (2024). Gpts are gpts: Labor market impact potential of llms. *Science*, *384*(6702), 1306–1308.

Epoch AI. (2026). *Epoch capabilities index.* https://epoch.ai/benchmarks. (AI Benchmarking Hub. Accessed November 14, 2026)

Erdil, E., & Besiroglu, T. (2023). *Explosive growth from AI automation: A review of the arguments.* arXiv preprint 2309.11690. Retrieved from https://arxiv.org/abs/2309.11690

Exner, Y., Hartmann, J., Ding, Z., Zhang, S., & Netzer, O. (2025, January). *AI in Disguise—Quasi-Experimental Analysis of a Large-Scale Deployment of AI-Generated Ads* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2026-07-19, from https://papers.ssrn.com/abstract=5096969 doi: 10.2139/ssrn.5096969

Farquhar, S. (2019, September). *Reaching new heights in the cloud.* Retrieved 2025-07-01, from https://www.atlassian.com/blog/platform/cloud-premium

Filippucci, F., Gal, P., & Schief, M. (2024). *Miracle or myth? Assessing the macroeconomic productivity gains from artificial intelligence* (OECD Artificial Intelligence Papers). OECD Publishing. doi: 10.1787/b524a072-en

Gans, J. S., & Goldfarb, A. (2026, January). *O-Ring Automation* [Working Paper]. National Bureau of Economic Research. Retrieved 2026-06-29, from https://www.nber.org/papers/w34639 doi: 10.3386/w34639

Ghose, A., & Han, S. P. (2014). Estimating demand for mobile applications in the new economy. *Management Science*, *60*(6), 1470–1488. doi: 10.1287/mnsc.2014.1945

Ghose, A., & Sundararajan, A. (2005). *Pricing security software: Theory and evidence* (Working Paper No. CeDER-05-23). Center for Digital Economy Research, Stern School of Business, New York University. Retrieved from https://archive.nyu.edu/handle/2451/14133 (Extended abstract presented at the Workshop on the Economics of Information Security (WEIS))

Gimbel, M., Kinder, M., Kendall, J., & Lee, M. (2025, October). *Evaluating the Impact of AI on the Labor Market: Current State of Affairs.* Retrieved 2026-03-12, from https://budgetlab.yale.edu/research/evaluating-impact-ai-labor-market-current-state-affairs

GitHub. (2025). *Compare GitHub to the competition.* Retrieved 2025-07-01, from https://resources.github.com/devops/tools/compare/

Goldberg, S., & Lam, H. T. (2025, February). *GENERATIVE AI & CREATIVE GOODS: MARKET EXPANSION, CROWD-OUT, AND COPYRIGHT* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2026-07-16, from https://papers.ssrn.com/abstract=5152649 doi: 10.2139/ssrn.5152649

Griffin, A., & Hauser, J. R. (1996, May). Integrating R&D and marketing: A review and analysis of the

literature. *Journal of Product Innovation Management*, *13*(3), 191–215. Retrieved 2026-07-16, from https://www.sciencedirect.com/science/article/pii/0737678296000252 doi: 10.1016/0737-6782(96)00025-2

Guha, A., Grewal, D., Kopalle, P. K., Haenlein, M., Schneider, M. J., Jung, H., … Hawkins, G. (2021, March). How artificial intelligence will affect the future of retailing. *Journal of Retailing*, *97*(1), 28–41. Retrieved 2026-07-19, from https://www.sciencedirect.com/science/article/pii/S0022435921000051 doi: 10.1016/j.jretai.2021.01.005

Hauser, J., Tellis, G. J., & Griffin, A. (2006, November). Research on Innovation: A Review and Agenda for Marketing Science. *Marketing Science*, *25*(6), 687–717. Retrieved 2026-07-16, from https://pubsonline.informs.org/doi/abs/10.1287/mksc.1050.0144 doi: 10.1287/mksc.1050.0144

Henderson, R. M., & Clark, K. B. (1990). Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly*, *35*(1), 9–30. doi: 10.2307/2393549

Hoffmann, M., Boysel, S., Nagle, F., Peng, S., & Xu, K. (2024). *Generative AI and the Nature of Work.* Retrieved 2024-11-07, from https://www.ssrn.com/abstract=5007084 doi: 10.2139/ssrn.5007084

Hull, P. (2026, October). *Two-Way Fixed Effects and Modern Difference-in-Differences: An Empirical Comparison.*

Humlum, A., & Vestergaard, E. (2025). *Large language models, small labor market effects* (Tech. Rep.). National Bureau of Economic Research.

IBM. (2024, January). *Data Suggests Growth in Enterprise Adoption of AI is Due to Widespread Deployment by Early Adopters.* Retrieved 2025-12-22, from https://newsroom.ibm.com/2024-01-10-Data-Suggests-Growth-in-Enterprise-Adoption-of-AI-is-Due-to-Widespread-Deployment-by-Early-Adopters

Iscenko, Z., & Millet, F. C. (2026, January). Looking for the Ladder: Is AI Impacting Entry-Level Jobs?

Jira. (2025). *Jira work items.* Retrieved 2025-07-03, from https://www.atlassian.com/software/jira/guides/issues/overview#what-is-an-work%20item

Johnston, D., Holtz, D., Richmond, A. M., Ong, C., Tambe, P., & Chatterji, A. (2026, June). *The Shift to Agentic AI: Evidence from Codex.* arXiv. Retrieved 2026-08-01, from http://arxiv.org/abs/2606.26959 (arXiv:2606.26959 [econ.GN]) doi: 10.48550/arXiv.2606.26959

Jones, B. (2025, October). *ARTIFICIAL INTELLIGENCE IN RESEARCH AND DEVELOPMENT.*

Jones, C. (2026, January). *A.I. and Our Economic Future* [Working Paper]. National Bureau of Economic Research. Retrieved 2026-06-29, from https://www.nber.org/papers/w34779 doi: 10.3386/w34779

Jones, C., & Tonetti, C. (2026, January). Past Automation and Future A.I.: How Weak Links Tame the Growth Explosion.

Jones, C. I. (2024). The AI dilemma: Growth versus existential risk. *American Economic Review: Insights*, *6*(4), 575–590. doi: 10.1257/aeri.20230570

Jordan, W. C., & Graves, S. C. (1995). Principles on the benefits of manufacturing process flexibility. *Management Science*, *41*(4), 577–594. doi: 10.1287/mnsc.41.4.577

Kessler, S. (2023). As artificial intelligence proliferates, the u.s. job market keeps up—but not for

everyone. *The New York Times*. Retrieved from https://www.nytimes.com/2023/06/10/business/ai-jobs-work.html (Business section)

Kim, A. G., Muhn, M., & Nikolaev, V. V. (2024, October). *From Transcripts to Insights: Uncovering Corporate Risks Using Generative AI* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-05-09, from https://papers.ssrn.com/abstract=4593660 doi: 10.2139/ssrn.4593660

Kinder, M., de Souza Briggs, X., Muro, M., & Liu, S. (2024, October). *Generative AI, the American worker, and the future of work.* Retrieved 2025-04-27, from https://www.brookings.edu/articles/generative-ai-the-american-worker-and-the-future-of-work/

Kingman, J. F. C. (1961). The single server queue in heavy traffic. *Mathematical Proceedings of the Cambridge Philosophical Society*, *57*(4), 902–904. doi: 10.1017/S0305004100036094

Klein Teeselink, B. (2025, September). *Generative AI and Labor Market Outcomes: Evidence from the United Kingdom* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2026-02-09, from https://papers.ssrn.com/abstract=5516798 doi: 10.2139/ssrn.5516798

Korinek, A., Jones, C. I., Sacher, S., Cotter, T., & McCrory, P. (2026, September). Economic Scenarios for Transformative AI.

Korinek, A., & Suh, D. (2024). *Scenarios for the transition to agi* (Tech. Rep.). National Bureau of Economic Research.

Kremer, M. (1993). The O-Ring Theory of Economic Development. *The Quarterly Journal of Economics*, *108*(3), 551–575. Retrieved 2026-06-29, from https://www.jstor.org/stable/2118400 doi: 10.2307/2118400

Kumar, S., Goel, D., Zimmermann, T., Houck, B., Ashok, B., & Bansal, C. (2024, 11). *Time warp: The gap between developers' ideal vs actual workweeks in an AI-driven era* (Tech. Rep.). Microsoft Research. Retrieved from https://www.microsoft.com/en-us/research/publication/time-warp-the-gap-between-developers-ideal-vs-actual-workweeks-in-an-ai-driven-era/ (Survey of 484 software developers at Microsoft, fielded June–July 2024)

Kwa, T., West, B., Becker, J., Deng, A., Garcia, K., Hasin, M., … Chan, L. (2025). Measuring ai ability to complete long software tasks. In *Advances in neural information processing systems (neurips).*

Kyriakopoulos, K., Hughes, M., & Hughes, P. (2015, August). The Role of Marketing Resources in Radical Innovation Activity: Antecedents and Payoffs. *Journal of Product Innovation Management*, *33*(4). Retrieved 2026-07-16, from https://onlinelibrary.wiley.com/doi/abs/10.1111/jpim.12285

Lenarduzzi, V., Saarimäki, N., & Taibi, D. (2019). The technical debt dataset. In *Proceedings of the fifteenth international conference on predictive models and data analytics in software engineering* (pp. 2–11).

Lichtinger, G., & Hosseini Maasoum, S. M. (2025, August). *Generative AI as Seniority-Biased Technological Change: Evidence from U.S. Résumé and Job Posting Data* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-10-21, from https://papers.ssrn.com/abstract=5425555 doi: 10.2139/ssrn.5425555

Little, J. D. C. (1961). A proof for the queuing formula: L = λ W. *Operations Research*, *9*(3), 383–387. doi: 10.1287/opre.9.3.383

Little, J. D. C. (2011). Little's law as viewed on its 50th anniversary. *Operations Research*, *59*(3),

536–549. doi: 10.1287/opre.1110.0940

Loch, C. H., & Terwiesch, C. (1998). Communication and uncertainty in concurrent engineering. *Management Science*, *44*(8), 1032–1048. doi: 10.1287/mnsc.44.8.1032

Lüders, C. M., Bouraffa, A., & Maalej, W. (2022). Beyond duplicates: Towards understanding and predicting link types in issue tracking systems. In *Proceedings of the 19th international conference on mining software repositories* (pp. 48–60).

MacCormack, A., Rusnak, J., & Baldwin, C. Y. (2006). Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, *52*(7), 1015–1030. doi: 10.1287/mnsc.1060.0552

Manzoor, E., Ascarza, E., & Netzer, O. (2025, November). *Learning When to Quit in Sales Conversations.*

Melitz, M. J., & Ottaviano, G. I. P. (2008). Market size, trade, and productivity. *The Review of Economic Studies*, *75*(1), 295–316. doi: 10.1111/j.1467-937X.2007.00463.x

Mrázová, M., & Neary, J. P. (2017). Not so demanding: Demand structure and firm behavior. *American Economic Review*, *107*(12), 3835–3874. doi: 10.1257/aer.20160175

Mu, J. (2015, August). Marketing capability, organizational adaptation and new product development performance. *Industrial Marketing Management*, *49*, 151–166. Retrieved 2026-07-16, from https://www.sciencedirect.com/science/article/pii/S0019850115001728 doi: 10.1016/j.indmarman.2015.05.003

Newey, W. K., & McFadden, D. (1994). Large sample estimation and hypothesis testing. In R. F. Engle & D. L. McFadden (Eds.), *Handbook of econometrics* (Vol. 4, pp. 2111–2245). Elsevier. doi: 10.1016/S1573-4412(05)80005-4

Nordhaus, W. D. (2007). Two centuries of productivity growth in computing. *The Journal of Economic History*, *67*(1), 128–159. doi: 10.1017/S0022050707000058

Nordhaus, W. D. (2021). Are we approaching an economic singularity? Information technology and the future of economic growth. *American Economic Journal: Macroeconomics*, *13*(1), 299–332. doi: 10.1257/mac.20170105

Noy, S., & Zhang, W. (2023, July). Experimental evidence on the productivity effects of generative artificial intelligence. *Science*, *381*(6654), 187–192. Retrieved 2024-10-03, from https://www.science.org/doi/10.1126/science.adh2586 (Publisher: American Association for the Advancement of Science) doi: 10.1126/science.adh2586

Ortu, M., Destefanis, G., Adams, B., Murgia, A., Marchesi, M., & Tonelli, R. (2015). The jira repository dataset: Understanding social aspects of software development. In *Proceedings of the 11th international conference on predictive models and data analytics in software engineering* (pp. 1–4).

Otis, N., Clarke, R., Delecourt, S., Holtz, D., & Koning, R. (2024, February). *The Uneven Impact of Generative AI on Entrepreneurial Performance* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-05-06, from https://papers.ssrn.com/abstract=4671369 doi: 10.2139/ssrn.4671369

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023, February). *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.* arXiv. Retrieved 2024-07-23, from http://arxiv.org/abs/2302.06590 (arXiv:2302.06590 [cs]) doi: 10.48550/arXiv.2302.06590

Pressman, R. S. (2010). *Software engineering: a practitioner's approach* (7th ed.). Dubuque, IA: McGraw-Hill.

Raj, M., & Seamans, R. (2019). AI, labor, productivity, and the need for firm-level data. In A. Agrawal, J. Gans, & A. Goldfarb (Eds.), *The economics of artificial intelligence: An agenda* (pp. 553–566). University of Chicago Press. doi: 10.7208/chicago/9780226613475.003.0022

Rapoport, G., Bicanic, S., & Talabi, M. (2025, May). *Survey: Generative AI's Uptake Is Unprecedented Despite Roadblocks.* Retrieved 2025-12-22, from https://www.bain.com/insights/survey-generative-ai-uptake-is-unprecedented-despite-roadblocks/ (Section: Brief)

Roldan-Mones, A. (2024, August). When GenAI increases inequality: evidence from a university debating competition.

Roose, K. (2025). For some recent graduates, the A.I. job apocalypse may already be here. *The New York Times.* Retrieved from https://www.nytimes.com/2025/05/30/technology/ai-jobs-college-graduates.html (Technology column)

Sajadieh, S., Fattorini, L., Perrault, R., Gil, Y., Parli, V., Santarlasci, L., … Weld, D. (2026, April). *The AI index 2026 annual report* (Tech. Rep.). Stanford, CA: AI Index Steering Committee, Institute for Human-Centered AI, Stanford University. (arXiv:2606.15708)

Sarkar, S. (2025, November). *AI Agents, Productivity, and Higher-Order Thinking: Early Evidence From Software Development* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. Retrieved 2025-11-20, from https://papers.ssrn.com/abstract=5713646 doi: 10.2139/ssrn.5713646

Sarkar, S., & Melas-Kyriazi, L. (2026, April). *Returns to Intelligence.*

Seamans, R., & Raj, M. (2019, May). Primer on artificial intelligence and robotics. *Journal of Organizational Design*, *8*(11). Retrieved 2026-07-20, from https://link.springer.com/article/10.1186/s41469-019-0050-0?utm_source=getftrŹutm_medium=getftrŹutm_campaign=getftr_pilotŹgetft_integrator=wiley

Smith, R. P., & Eppinger, S. D. (1997a). Identifying controlling features of engineering design iteration. *Management Science*, *43*(3), 276–293. doi: 10.1287/mnsc.43.3.276

Smith, R. P., & Eppinger, S. D. (1997b). A predictive model of sequential iteration in engineering design. *Management Science*, *43*(8), 1104–1120. doi: 10.1287/mnsc.43.8.1104

So, K. C., & Tang, C. S. (1995). Optimal operating policy for a bottleneck with random rework. *Management Science*, *41*(4), 620–636. doi: 10.1287/mnsc.41.4.620

Song, F., Agarwal, A., & Wen, W. (2024). The impact of generative ai on collaborative open-source software development: Evidence from github copilot. *arXiv preprint arXiv:2410.02091.*

Stanley, M. (2025, October). *More Software, More Developer Jobs.* Retrieved 2025-12-17, from https://www.morganstanley.com/insights/articles/ai-software-development-industry-growth

Sun, L., & Abraham, S. (2021, December). Estimating dynamic treatment effects in event studies with heterogeneous treatment effects. *Journal of Econometrics*, *225*(2), 175–199. Retrieved 2025-11-07, from https://www.sciencedirect.com/science/article/pii/S030440762030378X doi: 10.1016/j.jeconom.2020.09.006

The Anthropic Institute. (2026, June). *When AI builds itself: Our progress toward recursive self-improvement, and its implications.*

https://www.anthropic.com/institute/recursive-self-improvement. (Accessed 15 October 2026)

Thomke, S. H. (1998). Managing experimentation in the design of new products. *Management Science*, *44*(6), 743–762. doi: 10.1287/mnsc.44.6.743

Thomke, S. H. (2003). *Experimentation matters: Unlocking the potential of new technologies for innovation.* Boston, MA: Harvard Business School Press.

Trammell, P., & Korinek, A. (2023). *Economic growth under transformative AI* (Working Paper No. 31815). National Bureau of Economic Research. doi: 10.3386/w31815

U.S. Bureau of Labor Statistics. (2025, May). *Occupational Employment and Wage Statistics: Software Developers (15-1252), National Estimates.* https://data.bls.gov/oesprofile/?major_group=150000Źoccupation=151252. (Accessed October 28, 2026)

U.S. Census Bureau. (2025). *Statistics of u.s. businesses (susb) annual data tables by establishment industry: 2022.* U.S. Department of Commerce, Washington, DC. Table *U.S. & states, NAICS, detailed employment sizes* (`us_state_6digitnaics_2022.xlsx`). Retrieved from https://www.census.gov/data/tables/2022/econ/susb/2022-susb.html (Data year 2022; released April 10, 2025. Accessed September 21, 2026)

Weyl, E. G., & Fabinger, M. (2013). Pass-through as an economic tool: Principles of incidence under imperfect competition. *Journal of Political Economy*, *121*(3), 528–583. doi: 10.1086/670401

Yeverechyahu, D., Mayya, R., & Oestreicher-Singer, G. (2024). The impact of large language models on open-source innovation: Evidence from github copilot. *arXiv preprint arXiv:2409.08379*.

Yotzov, I., Barrero, J. M., Bloom, N., Bunn, P., Davis, S. J., Foster, K. M., … Wang, B. Z. (2026, February). *Firm Data on AI.*

Zychlinski, N., & Gurvich, I. (2026). The production of service: A workload view of complementarity and substitution. *Stochastic Systems*, *16*(2), 165–185. doi: 10.1287/stsy.2025.0111