

Artificial Intelligence in the Firm: Bottlenecks in Software Production

Fiona Chen

Harvard University

Job Market Paper

James Stratton

Harvard University

Current version: August 4, 2026

First version: January 7, 2026

Click [here](#) for current version

Abstract

We study the impacts of AI coding assistants and agents on software engineering work, using a novel proprietary dataset from an engineering analytics platform, covering 300 million work events — including GitHub coding activity, Jira issues, and Google Calendar events — across 718 firms. We use a staggered difference-in-differences design, exploiting variation in firm-level adoption timing of AI coding assistants and agents. Both technologies increase coding productivity. However, productivity gains do not fully pass through to changes in software output or employment. For AI agents, this incomplete pass-through reflects a bottleneck from code review: review times increase, a larger share of code updates require revisions, and reviews involve more comments. We develop a model of software production to structure these results, in which AI affects both productivity and quality of intermediate outputs, and in turn can generate a review bottleneck and limit pass-through.

Contact: fionachen@g.harvard.edu and jstratton@g.harvard.edu. We are indebted to Erik Brynjolfsson, Raj Chetty, Larry Katz, Mandy Pallais, and Jesse Shapiro for their guidance and support throughout this project. We are also grateful 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, Deirdre Bloome, Sydnee Caldwell, Bharat Chandar, Mihai Codreanu, Jessica Dai, Andy Haupt, Ruru Hoong, Anders Humlum, Alex Imas, Chad Jones, Arjun Ramani, Aakaash Rao, Alana Renda, and Neil Thompson; as well as seminar participants at the Berkeley Artificial Intelligence Research (BAIR) Lab, Google, Harvard Economics, Harvard Kennedy School, Opportunity Insights, the Machine Learning in Economics Summer Conference at the University of Chicago, MIT Computer Science and AI Laboratory (CSAIL), MIT FutureTech, Stanford Digital Economy Lab, Stanford Economics, 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

Descriptions of generative AI’s economic impacts often invoke both visions of abundance and concerns about widespread unemployment. Today’s most prominent AI leaders have not shied away from bold proclamations: for instance, Dario Amodei has touted the promise of “a country of geniuses in a data center” while warning of “an oncoming white collar bloodbath” (Amodei, 2024; VandeHei & Allen, 2025). Many have described AI as potentially the most transformative technology since electricity or the printing press (Ng, 2017; Stackpole, 2024; C. Jones, 2026).

How has AI affected worker productivity, firm output, and employment? Though there is substantial evidence that AI can raise productivity on specific tasks (Noy & Zhang, 2023; Peng et al., 2023; Brynjolfsson, Li, & Raymond, 2025), these gains may not translate into commensurate changes in firm output or employment. Production processes typically involve multiple complementary tasks that are differentially exposed to AI. As a result, output may be constrained by “bottleneck” tasks, the nature of tasks may evolve, and firms may need to reallocate labor accordingly (Kremer, 1993; Aghion et al., 2017; Brynjolfsson et al., 2018; Acemoglu & Restrepo, 2019; Autor, 2022; Acemoglu, 2024; B. Jones, 2025; Gans & Goldfarb, 2026; C. Jones & Tonetti, 2026).

We study the impacts of firm AI adoption on software engineering work — a context that has experienced significant advancements in AI capabilities and take-up of AI (Kinder et al., 2024; Eloundou et al., 2024). Early versions of AI “assistants” could provide workers with real-time code suggestions, directly in their coding interfaces or through chat interfaces. More recently, firms have begun utilizing “agentic” tools, which have the capacity to semi-autonomously complete tasks of significant length and complexity. We examine the impacts of adoption of both AI coding assistants and AI coding agents.

We use a novel proprietary dataset from Jellyfish (JF), a software firm whose main product is an analytics platform for companies to understand the activity of their engineering teams. Our data provides information on approximately 300 million work events of 700,000 workers across 700 firms, from January 2021 to March 2026. JF integrates data from five sources: (1) HR information; (2) code version control systems, such as GitHub, which provide information on engineers’ coding activity; (3) issue management systems, such as Jira, which indicate engineers’ assignment to and progress on tasks and projects; (4) scheduling tools, such as Google Calendar, which provide information on

meeting activity and time allocation; and (5) generative AI usage, for GitHub Copilot, Cursor, and Claude Code. The scope of our data allows us to observe the full software production pipeline, from intermediate measures of coding activity to final software output, as well as employment, and to tie changes in these outcomes to AI adoption.

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, early versus late and non-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.

We begin by studying the effects of generative AI tools on coding productivity, using three common measures of coding output: 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 small increases in code output — 12% for lines of code, 9% for commits, and 5% for pull requests — with only the effect on commits being statistically significant. AI agents lead to large and significant increases in all three measures: 30% for lines of code, 20% for commits, and 23% for pull requests. These estimates are consistent with, though on the lower end of 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).¹

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, it may not be deployed. We next ask whether these productivity gains translate into increases in firm-level software output, using two measures of output: resolution of Jira issues, which occurs once a body of coding work is deployed to production; and resolution of Jira epics, which represent completion of entire projects or features. We find small positive, but statistically insignificant, effects on both measures following adoption of AI assistants and AI agents. For agents, our estimates rule out an increase in output of larger than 12% of the baseline mean — well below the 30% individual-level productivity gain — suggesting that firm-level output does not

¹The lower estimate likely reflects the fact that we are using firm-level adoption as our treatment definition, rather than individual-level adoption — and thus, the estimate aggregates heterogeneity in take-up.

scale one-for-one with coding productivity. One concern is that AI adoption may lead to an increase in size or complexity of issues, such that the raw issue count would understate true output growth. We address this by using machine learning methods to predict the length of time it would take for an engineer to resolve an issue. We find no evidence of an effect on the average predicted length of each issue.

Finally, we study the effects of AI adoption on 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 overall employment — measured by the total number of workers indicating employment at a firm on LinkedIn — as well as engineering employment — measured by the number of active workers on Jellyfish. With both outcome measures, we cannot attribute significant employment changes to AI. For AI agents, we estimate a precise zero effect on overall employment, ruling out a decline greater than 2.9%; and a noisier zero effect on engineering employment, ruling out a decline greater than 13.8%. In both cases, these estimates rule out a full pass-through of the coding productivity gain to cuts in employment. With AI assistants, we estimate a precise zero effect on overall employment, but the effect on engineering employment is more ambiguous due to differential pre-trends in the assistant specification.

Taken together, our results show that while AI tools substantially raise coding productivity, 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 and firms’ organizational responses. 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 using a survey of 100 Prolific engineers, 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.

We find evidence of a bottleneck from the code review process: The average time to review a pull request increases by 49%, the share of pull requests with changes requested nearly doubles, and the number of comments per pull request increases by 35%. Firms also reallocate labor toward review activities: the share of workers performing code reviews increases by 14%. This bottleneck persists

following the adoption of AI code review tools. Although firms increasingly use AI to assist with code review, human reviewers continue to play a central role in the review process.

We develop a model of software production and AI adoption to interpret our empirical findings and their managerial and policy implications. In the model, software development consists of two complementary stages: code writing and code review. AI adoption affects the code-writing stage through two channels: it increases the productivity of coders while also changing the quality of draft code entering review. The first channel increases the volume of code requiring review, while the second changes the amount of review effort required per unit of code. Together, these forces shape the demand for code review and can generate bottlenecks in the software production process. However, the two channels have different implications for labor demand. Assuming that output is comparatively fixed, higher coding productivity reduces labor demand by lowering the amount of coding labor required to produce a given level of output. By contrast, changes in code quality affect the amount of downstream verification required, potentially increasing demand for review labor and shifting labor toward code review.

Related literature. Our project contributes to several literatures. First, a new but rapidly growing literature examines the empirical effects of AI on the labor market. Numerous papers have examined the impacts of AI on task-specific productivity, finding gains across a wide range of settings, including writing (Noy & Zhang, 2023), customer support (Brynjolfsson, Li, & Raymond, 2025), and more (Choi & Schwarcz, 2023; Dell’Acqua et al., 2023; Chen & Chan, 2024; Kim et al., 2024; Otis et al., 2024; Roldan-Mones, 2024; Dell’Acqua et al., 2025; Dillon et al., 2025; Manzoor et al., 2025). More recently, the literature on AI has turned to assessing the impacts of AI on firm output and employment. The empirical evidence on AI’s employment effects has been mixed thus far, with some papers 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, rather than individual AI usage. This allows us to capture the aggregate impacts of AI, accounting for incomplete take-up and heterogeneous utility across workers. Furthermore, it allows us to estimate not only a productivity effect, but also the associated effects on firm output, employment, and firm organization. Second, we provide estimates of these effects across generations

of AI coding tools, allowing us to examine how the impacts of AI evolve as the technology improves.

Second, a broader economics literature examines how productivity gains 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). However, 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; Berg et al., 2023; Agrawal et al., 2024). We contribute to this literature by demonstrating how firms adapt to AI adoption by changing the structure of the code review process and reallocating labor towards this bottleneck. Relatedly, a marketing literature examines how firms organize the process of transforming ideas into successful products, emphasizing the integrated roles of marketing capabilities and organizational structure (e.g., Griffin & Hauser, 1996; Hauser 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 this literature by providing insights into the effects of AI on the product development process in software engineering.

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 these papers, our main contribution is to examine the effects of AI adoption in a firm setting, in which features of the production process and organizational structure may shape effects. This allows us to more directly identify the nature of bottlenecks, and 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 AI, including details about the data classification and AI take-up. In Section 3, we describe the effects of AI on productivity, output, and employment. In Section 4, we provide evidence on production bottlenecks. In Section 5, we describe the model. In Section 6, we conclude.

2 Data and institutional context

This section summarizes the data used in our analysis, and provides background on technology firms and generative AI coding tools.

2.1 Data sources

Jellyfish data. 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 the engineers’ productivity, including in relation to their use of AI coding tools. To provide these analytics, JF integrates data on engineers’ activity from several sources. We make use of five 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 March 2026. There are 725,938 workers across 718 firms

(Appendix Table 1).

Software production typically follows well-defined and documented processes, consisting of four main stages (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.² 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, 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 Figure 2. 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 the progress on tasks and projects through status updates. When an issue is first created, it is marked as “To do”. Eventually, once a team decides to work on it, it gets assigned to a worker and marked as “In progress.” An engineer then writes code for the issue; this code is tracked by the version control systems discussed below. Once the code has been written, reviewed and 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;

²Jellyfish 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.

indeed, Jira’s product description states that tasks “typically represent individual work items such as 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 also industry-standard: Atlassian, Jira’s owner, claims that four-fifths of Fortune 500 firms use Jira (Farquhar, 2019). JF integrates data from several task management systems, including Jira and Microsoft Azure DevOps. Our analysis sample includes 68,281,845 issues, for an average of 6.04 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.³

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.⁴ We provide two examples of the interface for GitHub in Figure 3. 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 200,945,700 commits, for an average of 17.77 per worker-month. We observe the metadata associated with each commit, including the time of the commit, the identity

³Some 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.

⁴As 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.

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 49,050,030 pull requests, for an average of 4.34 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.

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 0.92 at the worker-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, on average, workers make 4.1 commits for every pull request.

Google Calendar data. For a subset of 213 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 vs 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 21,270,247 events, for an average of 14.10 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.

Revelio data. 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.

2.2 Description of analysis sample

Our analysis sample consists of the Jellyfish client firms that satisfy two restrictions. First, the firm must have data present throughout our analysis timeframe. Second, the firm must have opted in to its data being used for research.

We provide a description of the firms in our analysis sample in Table 2. The average firm employed 1,652 total workers (measured using the Revelio merge, SD 6,015) and 241 active engineers (measured as active workers in Jellyfish, SD 363) in January 2023, reflecting a right-skewed size distribution. Firms are also generally well established: the average firm is 19.3 years old as of January 2023 (SD 21.4).

The firms span a broad range of business models and industries. We classify each firm’s revenue engine using an LLM-based classification of its LinkedIn description into one of three categories: firms that sell a software licence directly (SaaS licence, 52% of firms); firms whose revenue is a cut of the activity they intermediate or carry rather than a licence fee, such as marketplaces or firms bearing their own balance-sheet risk (digital service operator, 30%); and firms that use software as an input into their own operations rather than as the product itself, such as a financial services firm that builds proprietary trading infrastructure (software as input, 14%).

We similarly classify each firm’s customer industry, collapsing into four categories. The largest share of firms build horizontal software that serves a general business function rather than a specific industry vertical (25%). Among firms serving a specific vertical, the most common are financial services and insurance (16%) and healthcare and life sciences (12%); the remaining firms (48%) span a long tail of other verticals, including media, retail, transport, government and education, real estate, industrials, 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. In total, we observe role information for 372,912 workers. Software engineers make up the majority of workers with role information (55%). The remainder are split across revenue roles (e.g., sales and marketing, 15%), administrative roles (e.g., legal and finance, 14%), product and design roles (e.g., product managers and user interface engineers, 9%), data science and research roles

(e.g., machine learning researchers, 6%), other non-software engineering disciplines (e.g., hardware roles, 0.7%), and chief executives (e.g., CTOs, 0.3%).

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. 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 of over 60 million Jira issues, we convert each issue’s text into embeddings using a Sentence-BERT model, 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 SBERT embeddings, using issues resolved in 2022 as the training set (approximately 2 million issues with non-missing cycle time). The trained model is then applied to all issues to produce a predicted log cycle time for every task in the sample, which we use as a continuous proxy for task complexity.

Validation of supervised learning method. We validate our supervised learning approach in Appendix Table 6. 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. Figure 4 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. The interaction is reactive: the tool responds to the developer’s current context but does not independently initiate or sequence actions. 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, for which Jellyfish provides API integrations, two of the most widely adopted coding assistants.

AI coding agents are tools that can semi-autonomously complete longer, multi-step tasks. Unlike assistants, which respond to a developer’s immediate input, agents accept a high-level task description and independently plan and execute a sequence of actions — they are able to ingest large amounts of context, then write code across multiple files, run tests, interpret error messages, and iterate until the task is complete. The developer reviews and approves the agent’s output, but is not required to guide each intermediate step. 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.

Finally, AI code review tools are tools that 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).

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 Section B. 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 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-se-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 between 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 business 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 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. 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.

Take-up rate. Figure 5 shows the rates of firm- and individual-level take-up of AI assistants and AI agents.

Panels A and B display firm-level take-up of AI agents and AI assistants over calendar time, respectively. In Panel A, the share of firms that have adopted any AI agent rises from near zero in October 2024 to over 95% by January 2026, with the steepest acceleration occurring around the releases of Cursor Agent (November 2024) and Claude Code (February 2025), marked by vertical dashed lines. Panel B shows that the adoption of AI coding assistants (GitHub Copilot or Cursor) began earlier, with take-up climbing from near zero in February 2023 to approximately 45% of firms by April 2024. We truncate the assistant take-up window at 16 months, which we view as capturing the main period of AI assistant diffusion, and to avoid overlap with the period of take-up of AI coding agents.

Panels C and D display individual-level take-up of AI coding agents and AI coding assistants, respectively, measured in months elapsed since the firm’s first adoption. Each line corresponds to a quintile of firms ranked by their end-of-period worker adoption rate. By month 12, the median firm (40–60th percentile) has an engineer-level agent adoption rate of approximately 20%; the corresponding figure for AI assistants at month 16 is approximately 40%. There is, however, substantial heterogeneity across firms: among firms in the top quintile, agent adoption reaches roughly 60% of engineers by month 12, compared to near zero for firms in the bottom quintile. For assistants, the spread is similarly wide, with top-quintile firms reaching approximately 65% adoption against roughly 10–15% in the bottom quintile.

3 Effects on productivity, output, and employment

We begin by documenting the effects of AI adoption on coding productivity, output, and employment.

3.1 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 timing of firm-level adoption of assistants and agents. We implement the following regression for firms f and months m :

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

Here, $\text{Activity}_{f,m}$ represents our outcome variables, which are measured at the firm-by-month level, such as number of Jira issues resolved per firm-month. Our main variable of interest is $\text{AI}_{f,m}^k = \mathbb{1}\{m - \text{AI}_f = k\}$, where AI_f is the first month of adoption, so $\text{AI}_{f,m}^k$ 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, such as industry, as well as month fixed effects δ_m to control for time-varying characteristics, such as macroeconomic conditions. Finally, we also include time-varying controls for baseline firm size $X_{f,m}$, motivated by imbalance in treatment take-up, as described in the next section. 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 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 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. 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, early- and late- or non-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 B (Rapoport et al., 2025; IBM, 2024).

We may be concerned that some factors shape both the timing of adoption as well as 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 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 B. 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. 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.

Qualitative interviews. Alongside the quantitative analysis, we conduct a series of qualitative interviews to validate the findings of the paper. We interview 17 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. We attribute quotes by role rather than by name, and we have lightly edited them for clarity. Appendix Section B reports the interview material in full.

3.2 Effects on coding productivity

First, we document the effects of AI adoption on coding productivity. Our dataset records three main forms of coding activity: (1) Lines of code added; (2) GitHub commits, which represent incremental updates to code; and (3) GitHub pull requests created, which represent completed drafts of code, submitted for review. For each outcome, we construct a firm-month panel and normalize by the number of active workers, yielding a per-worker monthly rate.

Figure 6 displays our estimated effects on coding activity. Panels A and B display event studies showing the estimated effects of AI agents (and assistants) on lines of code added per worker-month. Using our pooled estimator, we find a significant effect from agents of 1,494 lines per worker-month (s.e. = 583), or 30% of the baseline mean of 5,017. The effect for AI assistants is positive but insignificant (Pooled $\hat{\beta}$ = 420, s.e. = 371; baseline = 3,498).

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. To assess this, we also examine the effects on number of commits per worker-month in Panels C and D and pull requests per worker-month in Panels E and F. For commits, we find a significant effect from agents of 4.57 commits per worker-month (s.e. =

1.12), or 20% of the baseline mean of 22.58. The effect for AI assistants is also positive and significant (Pooled $\hat{\beta} = 1.40$, s.e. = 0.64; baseline = 14.84), representing a 9% increase relative to baseline.

Pull requests represent a coarser unit of work than commits. Because pull requests must be reviewed by a team member, changes in pull request activity should be less susceptible to artificial changes in individual behavior. We find a large and significant effect from agents of 1.22 pull requests per worker-month (s.e. = 0.29), or 23% of the baseline mean of 5.34. The effect for AI assistants is positive but insignificant (Pooled $\hat{\beta} = 0.19$, s.e. = 0.17; baseline = 3.78). Across these three measures of coding activity, we obtain similarly sized estimates.

Anecdotal accounts of software engineers’ experiences with AI 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.

Finally, a recent literature has shown that two-way fixed effects estimators may be inconsistent in the presence of treatment effect heterogeneity (Borusyak et al., 2024; de Chaisemartin & D’Haultfœuille, 2024; Callaway et al., 2021; Sun & Abraham, 2021). In turn, we examine the sensitivity of our estimates to different specifications in Appendix Figure 1. We do not observe significant differences across any of these specifications. Thus, for the rest of the paper, we implement the event studies using two-way fixed effects.

As an additional validation of our methodology, Appendix Figure 2 benchmarks our estimates of the productivity effects of AI coding assistants against those reported in the broader literature. Our estimates are broadly comparable, though on the lower end, in magnitude, lending confidence that the results are not driven by features specific to our sample or design.

3.3 Effects on output

Next, we examine the impacts of this productivity shock on firm-level output. As described in Section 2, software engineering teams use Jira to track work, providing two natural measures of output. The first is the number of Jira *issues* resolved per worker per month. Issues represent discrete units of engineering work — a bug fix, a new feature component, or a configuration change — that are considered complete when the associated code is deployed to production. The second is the number

of Jira *epics* resolved per worker per month. Epics are higher-level units that aggregate related issues into a coherent project or feature, and thus capture broader deliverables rather than individual tasks.

Figure 7 displays event study estimates for both measures. Panels A and B show the effects on issues resolved per worker-month for AI agents and AI assistants, respectively. In neither case do we find a statistically significant effect (agents: pooled $\hat{\beta} = 0.12$, s.e. = 0.17, baseline mean = 3.67; assistants: pooled $\hat{\beta} = 0.09$, s.e. = 0.10, baseline mean = 2.70). The 95% confidence intervals allow us to rule out effects larger than 12% of the baseline mean for agents and 11% for assistants. In particular, this rules out an effect on output of the same magnitude as our estimates for the effects on coding productivity. Panels C and D replicate the analysis for epics and similarly show no significant effect.

A concern with the raw issue count is that AI adoption may change the composition of tasks — for instance, if engineers shift towards tackling larger or more complex issues post-adoption, a flat count would understate the true increase 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 4 plots event studies for average issue size under both measures. We find no evidence of a compositional shift in either case (LLM label — agents: pooled $\hat{\beta} = -0.02$, s.e. = 0.04; assistants: pooled $\hat{\beta} = 0.02$, s.e. = 0.03), suggesting that the null output result is not an artifact of task redefinition.

3.4 Effects on employment

Finally, we turn to the effects on employment.

We display the aggregate results in Figure 8. We begin by assessing the effects of AI adoption on a firm’s total employment, as measured by the total number of LinkedIn profiles attached to the firm, in Panels A and B. This measure allows us to assess effects on both engineering and non-engineering teams. Both panels display precise zero effects (agents: Pooled $\hat{\beta} = 0.01$, s.e. = 0.02; assistants: Pooled $\hat{\beta} = 0.00$, s.e. = 0.01).

We next turn to the effects on employment of engineers, as tracked by the number of active workers in JF data each firm-month, in Panels C and D. We do this for two reasons. First, the shock of AI adoption may be sufficiently localized to coding teams that there is no broader impact on firm-wide employment. Second, LinkedIn data may be slow to update layoffs, making a direct measure of engineers preferable for detecting near-term effects. AI agents show a null pooled effect (Pooled $\hat{\beta} =$

−0.04, s.e. = 0.05). AI assistants show a small positive and marginally significant estimate (Pooled $\hat{\beta} = 0.07$, s.e. = 0.03). However, we are cautious to attribute a causal interpretation to this estimate, as the pre-period trends for the assistant specification show some upward drift, suggesting the positive estimate may partly reflect pre-existing growth trajectories among early adopters.

Finally, 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 Figure 9. We do not find significant effects of either AI agents or AI assistants on junior or senior workers (junior – agents: Pooled $\hat{\beta} = 0.02$, s.e. = 0.01; assistants: Pooled $\hat{\beta} = 0.00$, s.e. = 0.01; senior – agents: Pooled $\hat{\beta} = 0.00$, s.e. = 0.02; assistants: Pooled $\hat{\beta} = 0.00$, s.e. = 0.01).

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. 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.

4 Production bottlenecks

We observe significant effects of AI coding tools on coding productivity, 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, productivity gains in code writing may not pass through to firm-level output. In particular, we focus on a bottleneck from the code review process.

4.1 Effects on code review

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.

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 10. 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 42% 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% of their time writing code.

Figure 11 displays the impact of AI adoption on the length of the code review process, measured as the number of days between when a pull request is submitted and when it is merged. AI agents significantly lengthen the review process (Pooled $\hat{\beta} = 3.45$, s.e. = 1.60), a 49% increase relative to the baseline mean of 7.03 days. AI assistants have no significant effect on review length (Pooled $\hat{\beta} =$

-0.38, s.e. = 0.97; baseline = 9.34 days).

AI may impact the code 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, the code review process itself may change — e.g., a form of “new work” — either because the code written now is of worse quality, or because the standards of code review have changed.

Figure 12 provides evidence on changes in the review process itself. Panels A and B display the share of pull requests for which at least one reviewer submitted a formal request for changes. With AI agents, the share of PRs with changes requested increases by 0.12 (s.e. = 0.01) relative to a baseline mean of 0.13, nearly doubling the rate. With AI assistants, the effect is small and insignificant (Pooled $\hat{\beta} = 0.01$, s.e. = 0.01; baseline = 0.14). Panels C and D display the number of review comments per pull request — i.e., the intensive margin of number of changes requested. AI agents increase this measure by 0.58 comments per PR (s.e. = 0.11), a 35% increase relative to the baseline mean of 1.66. AI assistants have no significant effect (Pooled $\hat{\beta} = 0.05$, s.e. = 0.07; baseline = 1.98).

This evidence is consistent with qualitative accounts of changes in the code review process. For instance, one junior software engineer describes:

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.

Finally, it might be the case that the number of comments or changes requested increase mechanically if PRs are just getting larger in size. We assess this possibility in Appendix Figure 5. 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 agents (Pooled $\hat{\beta} = -67.99$, s.e. = 126.91; baseline = 1167.97) or for AI assistants (Pooled $\hat{\beta} = 66.07$, s.e. = 100.90; baseline = 1167.17).

Together, these results indicate that AI agents substantially intensify the code review process: reviews take longer, are more likely to result in change requests, and attract more reviewer comments. AI assistants show no significant impact on any of these dimensions.

4.2 Effects on allocation of labor

Given the bottleneck in code review documented above, a natural question is whether firms respond by reallocating workers towards reviewing code. Whether this occurs depends on the degree to which workers can flexibly move between coding and review tasks.

Figure 13 examines how AI adoption shifts the allocation of workers across coding and review tasks. Panels A and B display the share of workers in a firm-month who write code (i.e., make at least one commit). AI agents have no significant effect on the share of coders (Pooled $\hat{\beta} = 0.01$, s.e. = 0.01; baseline = 0.43), and neither do AI assistants (Pooled $\hat{\beta} = 0.01$, s.e. = 0.01; baseline = 0.38).

Panels C and D display the share of workers who perform code review (i.e., comment on or approve at least one pull request). AI agents significantly increase the share of workers engaged in code review (Pooled $\hat{\beta} = 0.04$, s.e. = 0.01; baseline = 0.29), a 14% increase relative to baseline. AI assistants also increase the share of reviewers, though by a smaller amount (Pooled $\hat{\beta} = 0.02$, s.e. = 0.01; baseline = 0.21), a 10% increase relative to baseline.

These results suggest that AI agents prompt firms to expand the set of workers engaged in code review, without reducing the share engaged in code writing. This is consistent with a model in which coding and review are complementary: increased coding throughput raises demand for review, and firms meet this demand at the extensive margin by drawing additional workers into the review process.

4.3 Usage of AI in code review

Finally, one concern may be that this code review bottleneck is a temporary artifact of the firm adjustment process, and it can 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.

Modern AI coding tools are technically capable of performing code review tasks — flagging bugs, suggesting improvements, and generating inline comments — in addition to writing code. Whether firms actually use them this way, however, is an empirical question. 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 14 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 March 2026, nearly 80% 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. 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 Model

We develop a model of technology adoption and firm production. The model serves two purposes. First, it provides a framework for interpreting our empirical results. Second, it helps illustrate implications for firm strategy. We develop the main ideas in this section, leaving the formal proofs to Appendix G.

A growing literature examines how task-specific productivity improvements from technology adoption translate into firm output and labor demand. A common insight from these models is that production consists of multiple complementary tasks, such that productivity gains in one task need not translate into proportional increases in firm output if other tasks serve as “bottlenecks” or limiting factors (Kremer, 1993; Aghion et al., 2017; B. Jones, 2025; Gans & Goldfarb, 2026; C. Jones & Tonetti, 2026). However, these models are largely agnostic about the specific mechanisms that generate these constraints. In practice, bottlenecks may arise for a number of reasons, including the sequential nature of production, quality issues, or liability concerns.

We develop a model to explore *why* bottlenecks may arise. In our model, production involves two stages. The second stage verifies the output of the first. AI adoption affects both productivity and output quality in the first stage. In turn, it can generate a bottleneck in the second stage, which limits firm output and shapes labor allocation. We write this model in the context of software production — where the two steps are code writing and code review. However, the same logic could apply to a broad range of contexts in which production involves both execution as well as verification or

managerial oversight.

5.1 Model set-up

Setting. A firm ships software output. Each unit of software output can be either good software G or a bug B . The firm earns revenue $R(G) - lB$, where R is increasing and concave, and l is the loss created by each bug.

Software production involves two sequential steps. First, coders write draft code. The quantity of draft code written is $Q = A(t)L_C$, where $A(t)$ is labor productivity and dependent on the technology t , and L_C is the number of coders. Each unit of draft code is a bug with probability $\pi(t)$ and good with probability $[1 - \pi(t)]$. Second, reviewers can assess code before shipping it. If a reviewer spends r units of time reviewing the code, the probability of detecting buggy code is $d(r)$, for an increasing, concave function d . Buggy code is thrown out and does not contribute to revenue.

Firm's problem. We simplify the firm's problem by assuming that (1) draws of bugs and bug detection are i.i.d. across units of code, and (2) the firm produces a large number of units of code. Since d is concave, the firm should spend the same amount of time r assessing each piece of code. As a result, after applying the law of large numbers, the firm's problem is:

$$\begin{aligned} \max_{L_C, L_R, r, Q} \quad & \underbrace{R(Q(1 - \pi(t)))}_{\text{Revenue from good code}} - \underbrace{lQ \pi(t) [1 - d(r)]}_{\text{Losses from bugs}} - \underbrace{(w_C L_C + w_R L_R)}_{\text{Wage bill}} \\ \text{s.t. (1) Total output of code: } & Q = A(t)L_C, \\ \text{(2) Total time reviewing code: } & L_R = rQ. \end{aligned}$$

For ease of notation throughout this section, we will define a few additional terms. First, define total labor as $L := L_C + L_R$ and good software output as $G := Q(1 - \pi)$. Then, define the marginal cost of producing one unit of draft code, evaluated at the optimal review intensity as $c^* := \frac{w_C}{A} + w_R r^* + l\pi[1 - d(r^*)]$. Accordingly, define the cost shares of code writing $s_C := \frac{w_C/A}{c^*}$, reviewing $s_R := \frac{w_R r^*}{c^*}$, and bugs $s_B := \frac{l\pi[1 - d(r^*)]}{c^*}$. Finally, define the curvature of marginal revenue $\rho := -d \ln R'(G^*)/d \ln G^* > 0$ and curvature of marginal bug detection $\lambda := -d \ln d'(r^*)/d \ln r^* > 0$.

Proposition 1 (Firm's optimum). *The firm's optimal choices (L_C^*, L_R^*, r^*, Q^*) are characterized by the following expressions:*

1. *Review time*: $d'(r^*)l\pi(t) = w_R$.
2. *Code quantity*: $[1 - \pi(t)]R'(Q^*(1 - \pi(t))) = c^*$.
3. *Coding labor*: $L_C^* = \frac{Q^*}{A(t)}$.
4. *Non-coding labor*: $L_N^* = r^* Q^*$.

We leave the proof of this proposition, as well as those that follow, to Appendix Section G. To see the intuition behind these expressions: the first equation indicates that the optimal time for reviewing each piece of code r^* equates the marginal benefit of an extra bit of code with the marginal cost. The marginal benefit is determined by the likelihood of finding another error, multiplied by the loss avoided per error. The marginal cost is the wage bill from additional review time. The second equation indicates that the optimal quantity of draft code to produce Q^* similarly equates the marginal benefit — the probability that the code is bug free, multiplied by the marginal revenue it generates — with the marginal cost. The optimal allocation of labor to coding and review follow from their definitions.

5.2 Effects of AI adoption

AI adoption can affect the software production process through two channels. First, it can affect the *productivity of code writers*, $A(t)$. Second, it can affect the *bug rate*, $\pi(t)$. In this section, we characterize the effects of these two channels.

Proposition 2 (Effects of increase in coding productivity). *An increase in coding productivity $A(t)$ has the following implications.*

1. *No change in review time per unit of code:*

$$\frac{d \ln r^*}{d \ln A} = 0$$

2. *Increase in draft code and good code:*

$$\frac{d \ln Q^*}{d \ln A} = \frac{d \ln G^*}{d \ln A} = \frac{s_C}{\rho}$$

3. *Ambiguous effect on coding employment:*

$$\frac{d \ln L_C^*}{d \ln A} = \frac{s_C}{\rho} - 1$$

4. *Increase in review employment:*

$$\frac{d \ln L_R^*}{d \ln A} = \frac{d \ln Q^*}{d \ln A}$$

To see the intuition, note first that the change in coding productivity does not affect the firm’s “inner” problem of optimally reviewing code; as a result, the optimal review time r^* is unchanged (as in equation 1). On the other hand, as the firm becomes more productive, it tends to increase its output of code (as in 2). The scale of this output is mediated by (a) the extent of diminishing marginal returns to extra code ρ , and (b) the cost share of code s_C . The firm will employ more reviewers, since the review time per unit is unchanged, and more code is produced (as in 4). On the other hand, whether the firm employs more or fewer coders is ambiguous (as in 3): intuitively, there is both a *substitution effect*, causing the firm to require fewer coders per unit of output, and a *scale effect*, causing the firm to expand its coding output.

Proposition 2 assumes that the firm can freely reallocate labor between coding and review as its productivity changes. In practice, this reallocation may be slow or costly: workers may lack the skills to shift into review, or firms may be slow to recognize the importance of this reallocation. In turn, this may generate a bottleneck effect. Thus, we next characterize the consequences of AI adoption absent this reallocation.

Proposition 3 (Bottleneck from productivity channel). *Consider the case that labor has not been reallocated, so review labor is fixed at $\bar{L}_R$. Evaluated at a point where $\bar{L}_R = r^*Q$, an increase in coding productivity $A(t)$ leads to a smaller increase in the amount of good software that ships than under flexible reallocation:*

$$\left. \frac{d \ln G^*}{d \ln A} \right|_{\bar{L}_R} = \left. \frac{d \ln Q^*}{d \ln A} \right|_{\bar{L}_R} = \frac{s_C}{\rho + \lambda s_R} < \frac{s_C}{\rho} = \left. \frac{d \ln G^*}{d \ln A} \right|_{flexible}.$$

Without additional reviewers, an increase in coding productivity still leads the firm to write more draft code, but because a fixed pool of reviewers must spread the same total review capacity over a growing volume of code, a smaller share of that additional code makes it through review without generating additional expected losses from undetected bugs. As a result, the amount of good software that actually ships rises by less than it would under flexible reallocation. The size of this shortfall, captured by λs_R , is larger when detection exhibits more sharply diminishing returns to review time

(λ large) and when review already accounts for a larger share of costs (s_R large). This is the formal sense in which code review can act as a genuine bottleneck: absent reallocation of labor towards review, gains in coding productivity pass through less fully into completed, good software.

We have characterized the effects of the productivity channel, both with and without labor reallocation. AI tools may also negatively affect the quality of code written. This may occur for a number of reasons: AI tools may be faulty; additionally, coders may be less capable of catching bugs in AI-generated code than their own code. Thus, we next turn to characterizing the implications of an increase in the bug rate.

Proposition 4 (Effects of increase in the bug rate, bottleneck from bug rate channel). *An increase in the bug rate $\pi(t)$ has the following implications.*

1. *Increase in review time per unit of code:*

$$\frac{d \ln r^*}{d \ln \pi} = \frac{1}{\lambda}$$

2. *Decrease in good code:*

$$\frac{d \ln G^*}{d \ln \pi} = -\frac{1}{\rho} \left(s_B + \frac{\pi}{1 - \pi} \right)$$

3. *Ambiguous effects on draft code:*

$$\frac{d \ln Q^*}{d \ln \pi} = -\frac{1}{\rho} \left(s_B + \frac{\pi}{1 - \pi} \right) + \frac{\pi}{1 - \pi}$$

4. *Ambiguous effects on coding and review employment:*

$$\frac{d \ln L_C^*}{d \ln \pi} = \frac{d \ln Q^*}{d \ln \pi} \quad \text{and} \quad \frac{d \ln L_R^*}{d \ln \pi} = \frac{d \ln Q^*}{d \ln \pi} + \frac{1}{\lambda}$$

From this proposition, we see how a change in the bug rate can also generate a bottleneck. A higher bug rate leads to an increase in the amount of scrutiny required per unit of code r^* (equation 1), and reduces the amount of good software that ships G^* falls (equation 2). This stands in contrast to an increase in coding productivity, which unambiguously raises good output (Proposition 2). The total quantity of *draft* code — and the total employment of coders — could still rise or fall (as in equations 3 and 4): the firm may choose to produce *more* draft code, in order to ensure it has a

sufficiently large amount of code that *works*; on the other hand, each piece of code is more likely to produce bugs, so the firm may reduce its code output.

Finally, we characterize the *overall* effects of AI, combining the effects from the productivity and bug rate channels.

Corollary 1 (Overall effects of AI). *As AI affects both coding productivity $A(t)$ and the bug rate $\pi(t)$, its overall effects are characterized by the following.*

1. *Effect on review time per unit of code:*

$$\frac{d \ln r^*}{d \ln t} = \frac{1}{\lambda} \frac{d \ln \pi}{d \ln t}$$

2. *Effect on good output:*

$$\frac{d \ln G^*}{d \ln t} = \frac{1}{\rho} \left[s_C \frac{d \ln A}{d \ln t} - \left(s_B + \frac{\pi}{1 - \pi} \right) \frac{d \ln \pi}{d \ln t} \right]$$

3. *Effect on draft code:*

$$\frac{d \ln Q^*}{d \ln t} = \frac{d \ln A}{d \ln t} \frac{s_C}{\rho} + \frac{d \ln \pi}{d \ln t} \left[-\frac{1}{\rho} \left[s_B + \frac{\pi}{1 - \pi} \right] + \frac{\pi}{1 - \pi} \right]$$

4. *Effect on employment:*

$$\begin{aligned} \frac{d \ln L_C^*}{d \ln t} &= \frac{d \ln A}{d \ln t} \left[\frac{s_C}{\rho} - 1 \right] + \frac{d \ln \pi}{d \ln t} \left[-\frac{1}{\rho} \left[s_B + \frac{\pi}{1 - \pi} \right] + \frac{\pi}{1 - \pi} \right] \\ \frac{d \ln L_R^*}{d \ln t} &= \frac{d \ln A}{d \ln t} \frac{s_C}{\rho} + \frac{d \ln \pi}{d \ln t} \left[-\frac{1}{\rho} \left[s_B + \frac{\pi}{1 - \pi} \right] + \frac{\pi}{1 - \pi} + \frac{1}{\lambda} \right] \end{aligned}$$

This follows directly from Propositions 2 and 4. AI's overall impacts depend on the relative size of its impacts through coding productivity and the bug rate. To see how a large shock to coding productivity $A(t)$ can have a small impact on software output, consider equation 2. Here, the pass through is governed by two expressions. First, the productivity shock is scaled by both the cost share of coding s_c and the inverse of the revenue curvature ρ — i.e., pass through can be low if either s_c is low or ρ is high. Secondly, it can also be offset by the effect on the bug rate — a larger effect of AI adoption on π or a larger cost share of bugs would also result in a lower pass-through rate. Similarly, equation 4 demonstrates how the productivity shock may not entirely pass through to a change in employment.

Although the effects on coding and review employment are ambiguous, the effects on relative employment are not. We characterize this next.

Corollary 2 (Relative allocation of labor between coding and review). *The effect of AI adoption on relative review-to-coding employment is positive.*

$$\frac{d \ln(L_R^*/L_C^*)}{d \ln t} = \frac{d \ln A}{d \ln t} + \frac{1}{\lambda} \frac{d \ln \pi}{d \ln t}.$$

This follows immediately from Proposition 1: since $L_C^* = Q^*/A$ and $L_R^* = r^*Q^*$, the quantity of draft code Q^* cancels from the ratio entirely, leaving a relationship that depends only on review intensity and coding productivity, not on the curvature of revenue or any cost share. Because r^* does not respond to A (Proposition 2) but rises with π (Proposition 4), the coding-productivity channel and the bug-rate channel never offset each other in this ratio the way they can for L_C^* or Q^* individually: both channels act to increase it. A firm's staffing mix therefore shifts toward review — and never toward coding — as AI adoption raises coding productivity or the bug rate. This provides the cleanest theoretical counterpart to our finding that AI adoption raises the share of workers engaged in code review without reducing the share who write code.

Finally, to separately demonstrate the importance of the productivity and bug rate channels, we consider the limiting case where $\rho \rightarrow \infty$ — i.e., the revenue function is very concave — and hence, the firm passes through less of the coding productivity shock to output.

Proposition 5 (Employment effects under limited output pass-through). *Consider the limiting case of Corollary 1 in which $\rho \rightarrow \infty$. Then, for coding labor, the coding-productivity channel becomes purely labor-displacing, so the overall employment effect is determined by the relative magnitudes of the productivity and bug-rate channels. For review labor, the employment effect is determined entirely by the bug-rate channel. In particular,*

$$\begin{aligned} \frac{d \ln L_C^*}{d \ln t} &= -\frac{d \ln A}{d \ln t} + \frac{d \ln \pi}{d \ln t} \frac{\pi}{1 - \pi} \\ \frac{d \ln L_R^*}{d \ln t} &= \frac{d \ln \pi}{d \ln t} \left[\frac{\pi}{1 - \pi} + \frac{1}{\lambda} \right] \end{aligned}$$

This proposition describes a case in which the coding productivity and bug rate channels have

distinct effects on employment. As $\rho \rightarrow \infty$, the revenue function becomes highly concave, so firms no longer respond to higher coding productivity by expanding output. In other words, the scale effect disappears. The coding productivity channel therefore becomes purely labor displacing: higher coding productivity reduces the demand for coders and has no direct effect on review labor. Consequently, any increase in employment must arise through the bug rate channel. Intuitively, if AI-generated code requires additional verification, firms reallocate labor toward reviewing and debugging, creating new verification work that can offset the labor-saving effects of higher coding productivity.

5.3 Discussion

The model provides a useful framework for interpreting our empirical results. Empirically, we find that AI agents increase coding activity, review time, the share of pull requests with changes requested, and the number of comments per pull request (Figures 11 and 12). Through the lens of the model, the increase in coding activity identifies the productivity channel, while the increase in changes requested and comments per review — unexplained by productivity gains alone — identifies the bug-rate channel.

Together, these findings help explain why AI can substantially increase coding activity while generating much smaller effects on software output and employment. As shown in Corollary 1 above, the ultimate effect on completed software output depends on the net effect of the productivity and quality channels, scaled by $1/\rho$: productivity improvements encourage firms to produce more draft code, while increased verification requirements limit the extent to which these gains pass through to completed software. The earlier propositions further demonstrate how the productivity and quality channels each separately shape the output and employment effects.

These results and the model have important implications for managers and policymakers. Realizing the benefits of AI adoption requires complementary investments in downstream review capacity, rather than simply increasing code output. Furthermore, the appropriate firm response also depends on the channel through which AI affects production. For instance, if AI solely affects production through coding productivity, then labor reallocation can alleviate much of the bottleneck effect. However, if the effect occurs through the quality channel, then alleviation would require improvements in the technology itself.

6 Conclusion

Generative AI has garnered significant attention in the software context, yet relatively little is known about how improvements in individual productivity translate into firm-level outcomes. Using novel data from an engineering analytics platform, we study the effects of firm adoption of AI coding assistants and agents on software production. We find that both technologies increase coding productivity, with larger effects for AI agents. Despite these gains, we find little evidence that firms increase software output or reduce employment. Instead, the productivity gains are absorbed by downstream constraints in the production process.

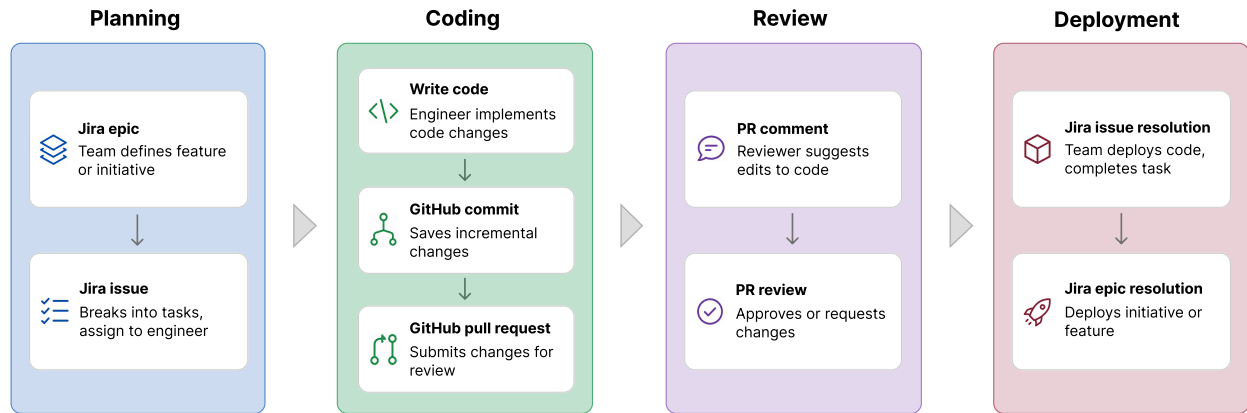
In particular, code review becomes the bottleneck. Following adoption of AI agents, the code review process significantly increases in lengths, pull requests are more likely to require revisions, and reviewers leave more comments. Firms respond by reallocating labor towards code review. This bottleneck is not simply alleviated through adoption of AI code review tools — even after adoption, humans are still involved in the code review process. Our model structures these findings by showing that AI can have dual effects on production, by changing coding productivity as well as the bug rate of code.

These findings have several implications for managers. First, measuring the returns to AI requires evaluating the entire production process. Many managers rely on intermediate productivity metrics, such as lines of code or commits; however, these may not accurately capture the true gains from AI adoption. Second, realizing the benefits of AI adoption requires complementary organizational investments. Changes in one production stage may shift the bottleneck to another, and hence necessitate reallocation of labor.

More broadly, our findings suggest that the economics of AI adoption cannot be understood by studying automated tasks in isolation. Production processes consist of multiple complementary activities, and accelerating one stage often shifts the constraint elsewhere. Understanding where these bottlenecks arise — and how firms reorganize to address them — will be central to understanding the long-run effects of AI on productivity, organizational design, and labor demand.

A Figures

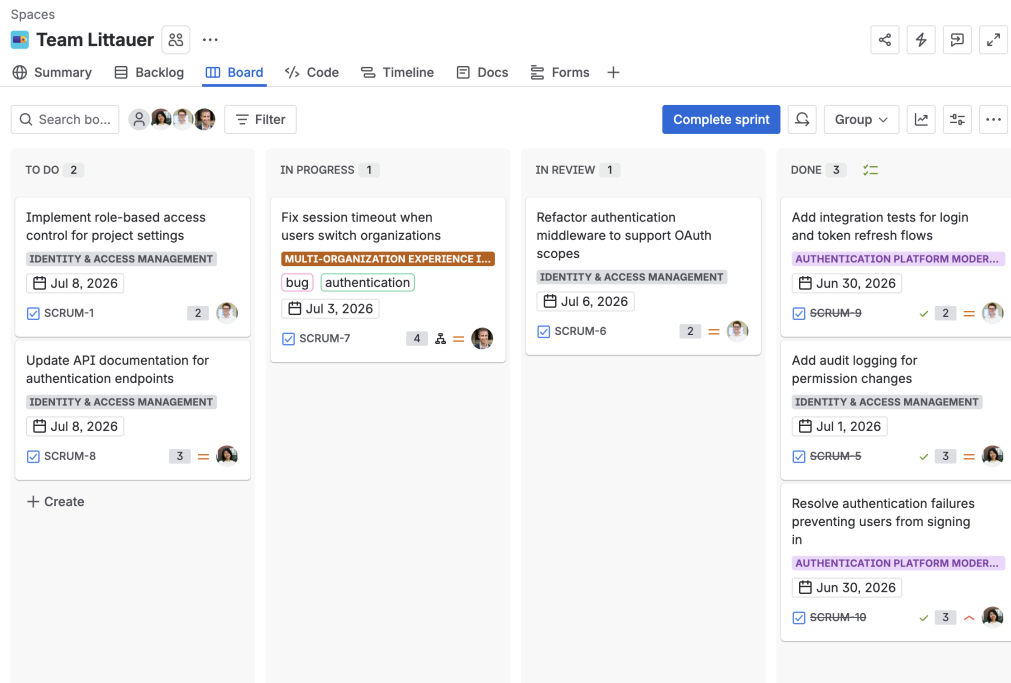
FIGURE 1. Software production process



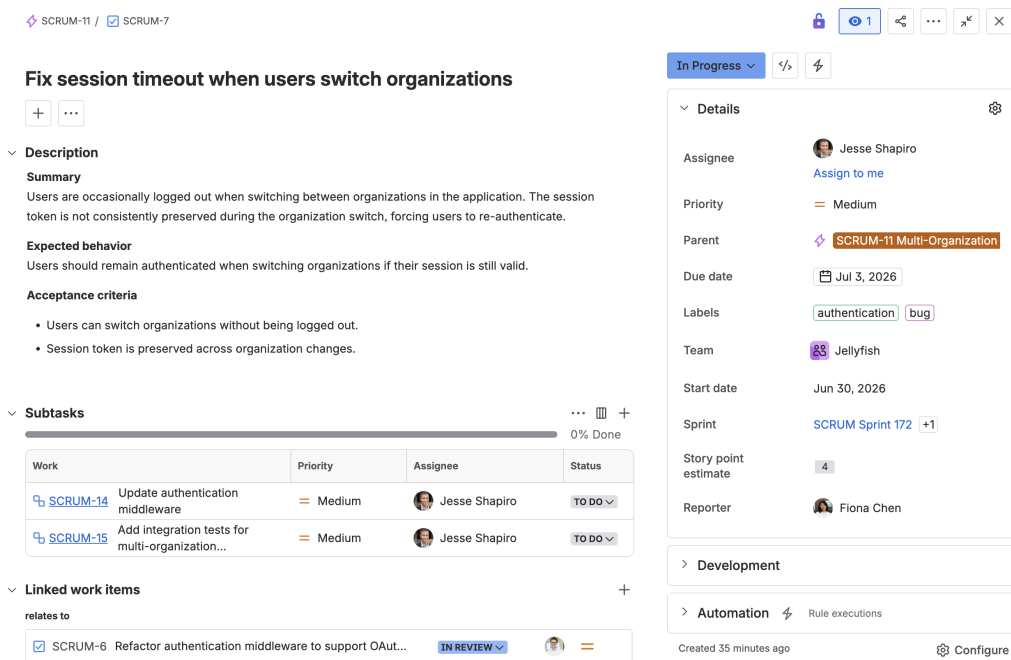
Notes: This figure provides an overview of the software production process, including a description of how our observed work signals fit into the production process.

FIGURE 2. Examples of Jira interface

A. Jira board



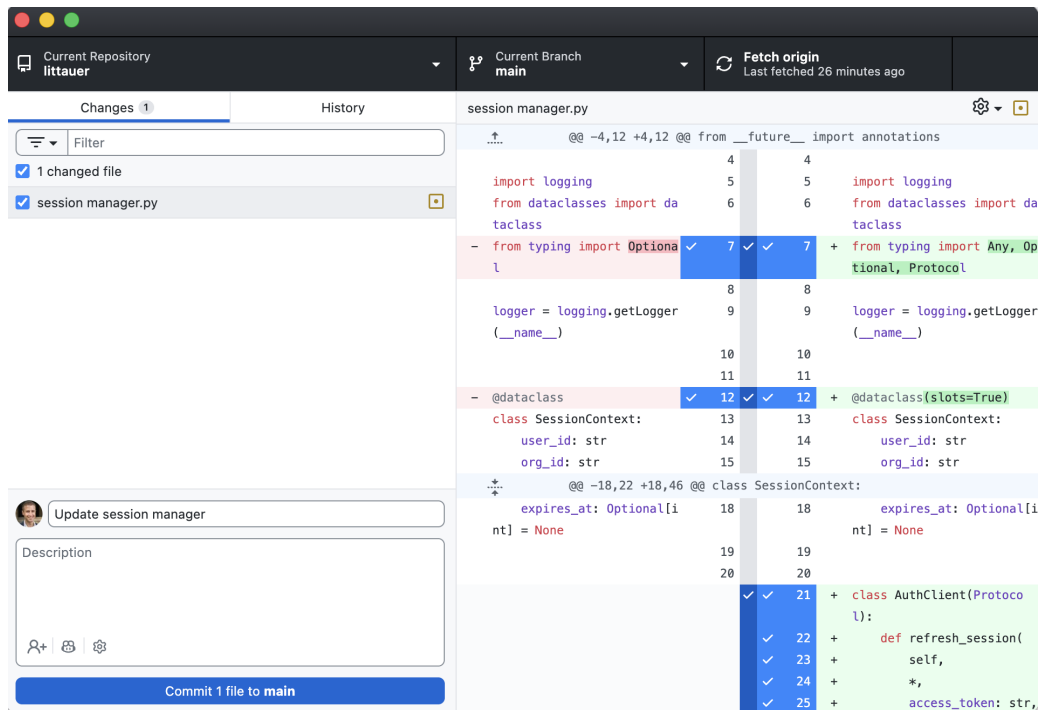
B. Jira issue



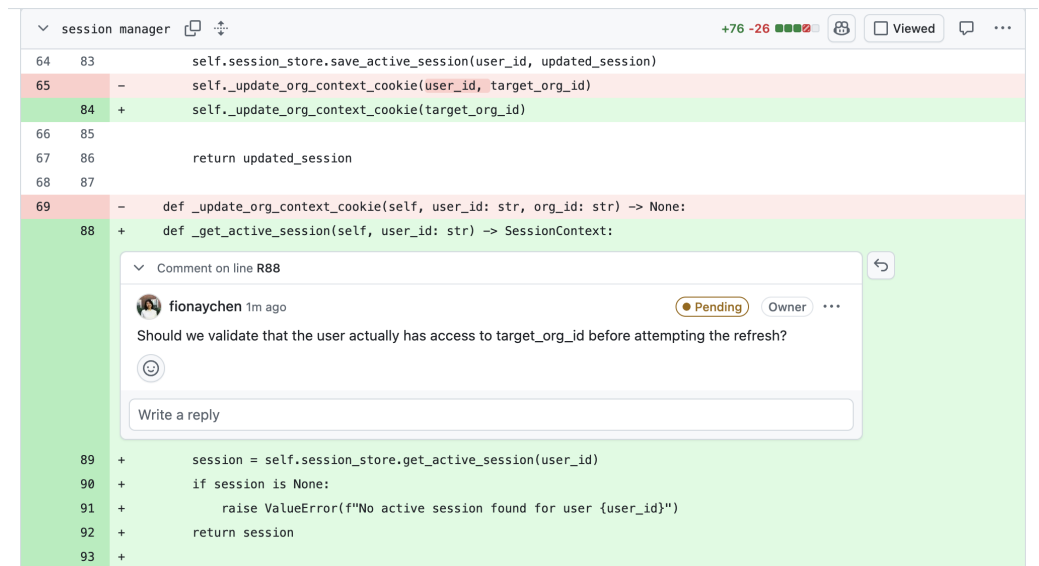
Notes: This figure displays two examples of the Jira interface. Panel A shows a team's Jira board, where software development work is organized into issues and tracked across workflow stages. Panel B shows the interface for an individual Jira issue, including its description, subtasks, linked epic, linked work items, and assignee.

FIGURE 3. Examples of GitHub interface

A. GitHub commit



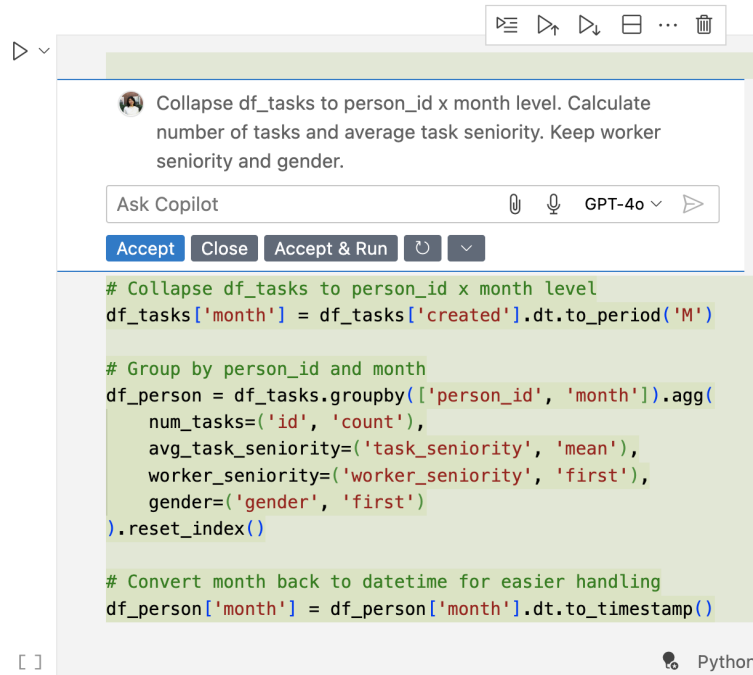
B. GitHub pull request review



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 4. Examples of AI coding tool interface

A. AI coding assistant



B. AI coding agent

I would like to write a do-file to clean the raw Jira task files. Please write a Python script that will do the following:

1. Import all the individual raw files, and merge them to the Jira user file.
2. Extract the embeddings from the text descriptions using sentence-transformers.
3. Use the Pytorch model to classify tasks according to coding vs non coding and task length.

The final file should be collapsed to the person_id x month level, with outcome variables for: tasks (count), cycle_time (mean), tasks_coding, tasks_noncoding, tasks_short, tasks_long.

- **Explore**(Explore Jira raw files and existing patterns)
 - Done (44 tool uses · 61.0k tokens · 1m 45s) (ctrl+o to expand)
- Now let me read the key existing notebooks to understand the exact patterns used before designing the plan.
 - Read 2 files (ctrl+o to expand)
- Now let me check how many chunk files exist in the current vintage and look at an existing clean script for the final collapse pattern:
 - Searched for 1 pattern, read 1 file, listed 1 directory (ctrl+o to expand)
- I have enough context. Let me write the plan.
- **Updated plan**
 - /plan to preview
- User approved Claude's plan
 - Plan saved to: ~/.claude/plans/temporal-sparking-waterfall.md · /plan to edit

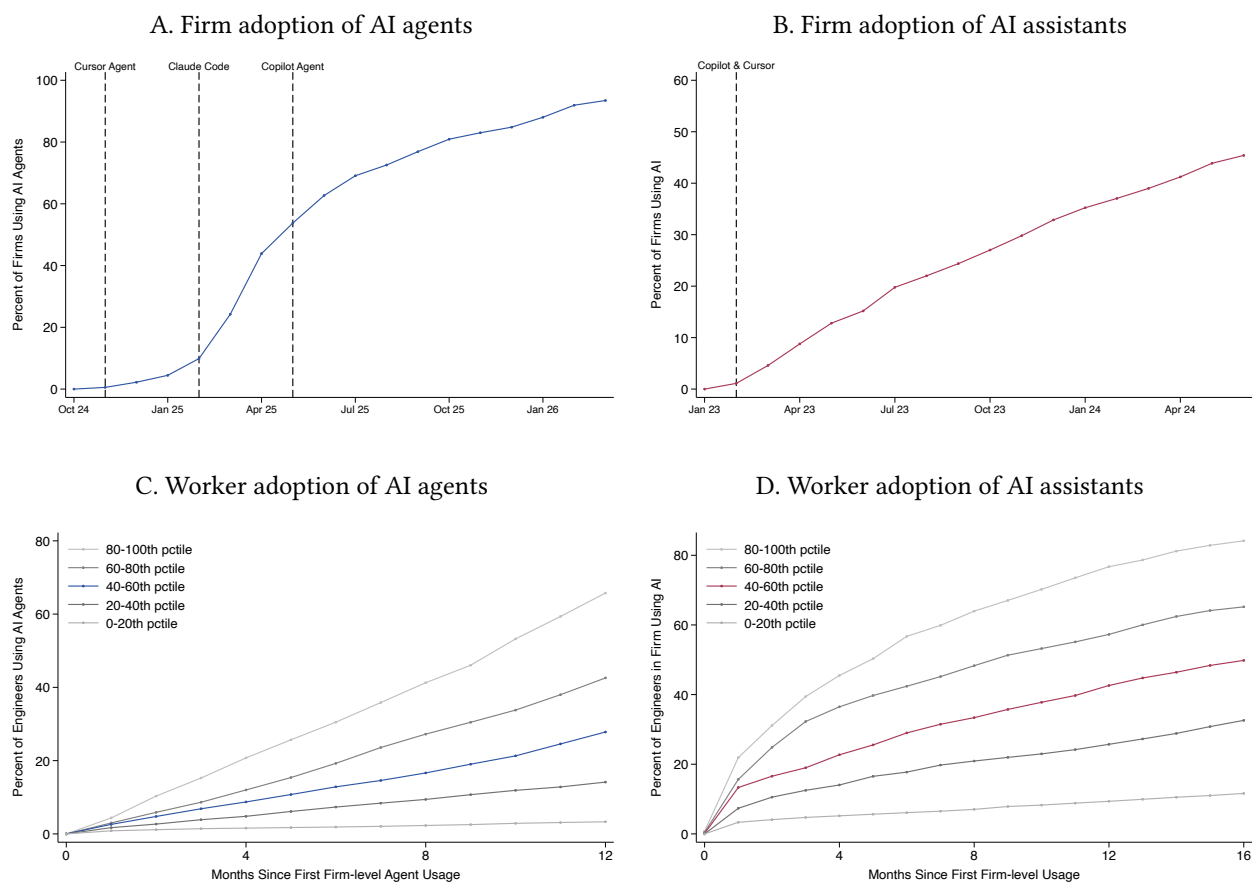
Plan: Clean Jira Tasks – Embeddings, Classification, Collapse

Context

We need a single Python notebook that takes the raw Jira issue chunk files, merges them with the Jira user file (to get `person_id`), generates sentence-transformer embeddings, runs the existing PyTorch models to classify tasks as coding/non-coding and short/long, and collapses the result to a `person_id x month` panel with the required outcome variables. This mirrors the existing `pytorch predict coding and length.ipynb` workflow but consolidates the full pipeline into one clean script targeted at the current vintage.

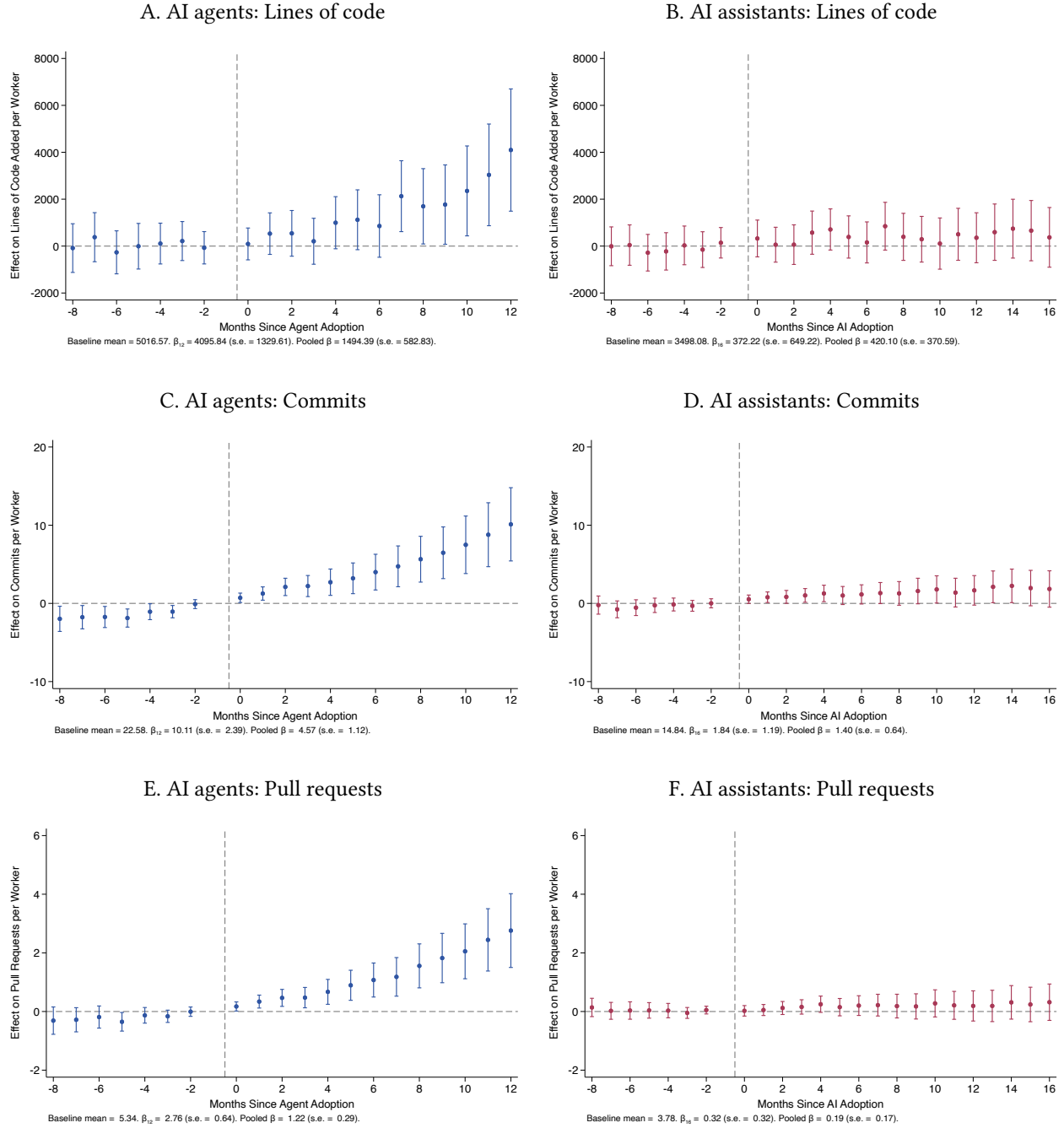
Notes: This figure displays two examples of the user interfaces for AI coding tools. Panel A displays the interface for an AI coding assistant, GitHub Copilot. Here, the user prompts the assistant, and the AI tool generates an individual block of code. Panel B displays the interface for an AI coding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

FIGURE 5. Adoption of generative AI tools



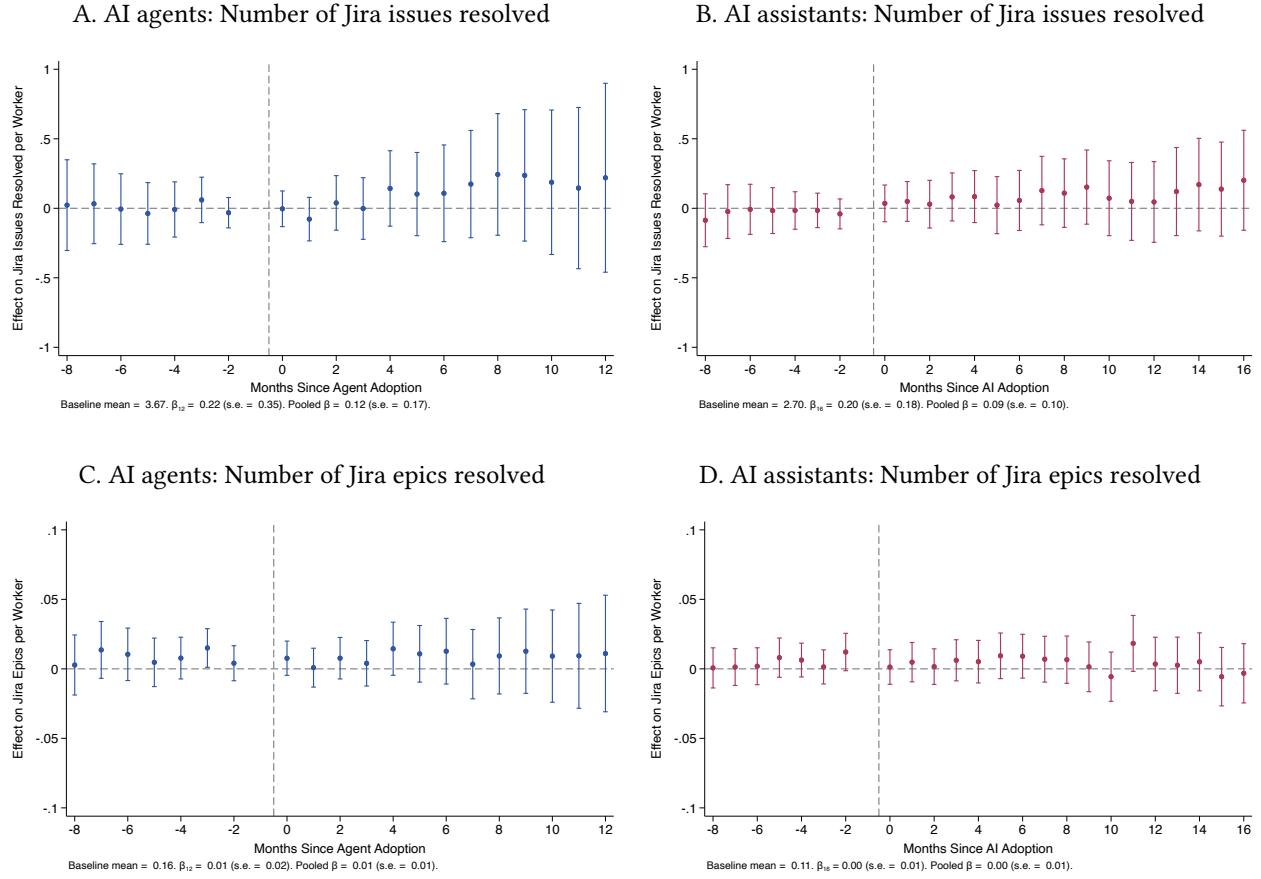
Notes: This figure displays information on firm-level and worker-level adoption of AI coding assistants and AI agents. Panel A displays firm-level adoption of AI agents over time, beginning in October 2024. Vertical lines mark the release of agentic features in Cursor (November 2024), Claude Code (February 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. Panel B 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. Panels C and D display worker-level adoption of AI agents and AI assistants respectively, in the months elapsed since the firm’s first adoption. The denominator is workers active on GitHub during the 12-month (Panel C) or 16-month (Panel D) window following firm adoption. Lines correspond to quintiles of firms ranked by their end-of-period worker adoption rate.

FIGURE 6. Effects of AI on coding activity



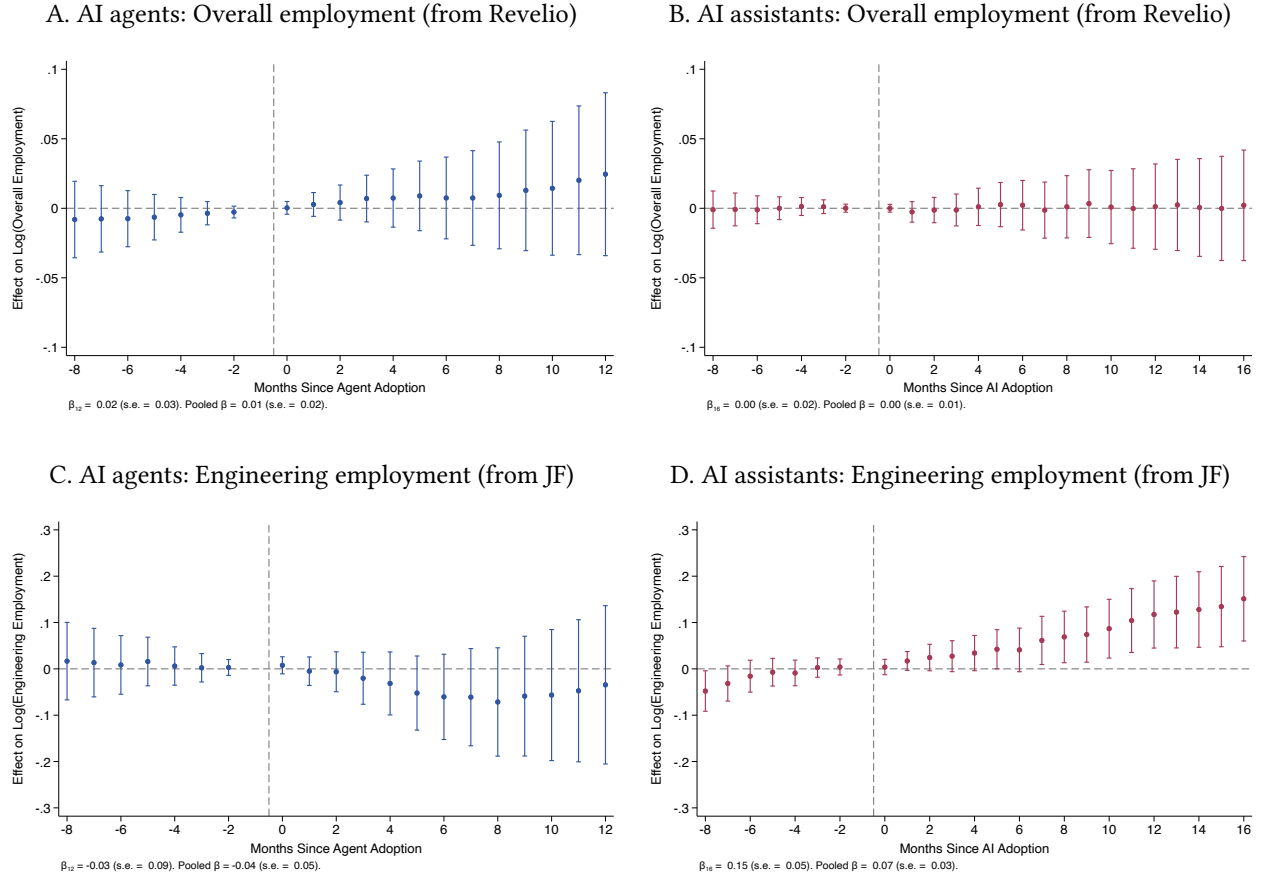
Notes: This figure displays event studies of the effects of AI adoption on coding productivity. In Panels A and B, the outcome variable is the number of lines of code added, normalized by the number of workers. We construct a firm-by-month panel, where the outcome variable in each row is the number of 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 coding agents. In Panels B, D, and F, 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 7. Effects of AI on 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 firm-level adoption of AI coding agents. In Panels B and D, 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.

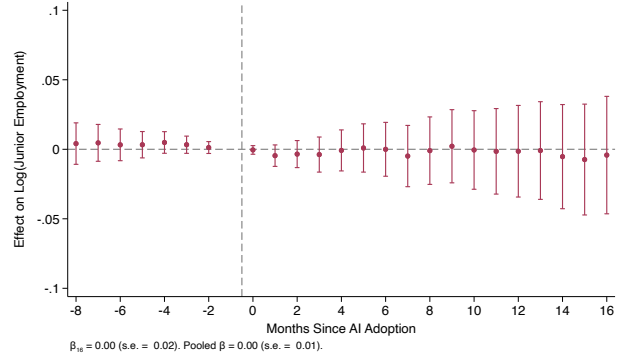
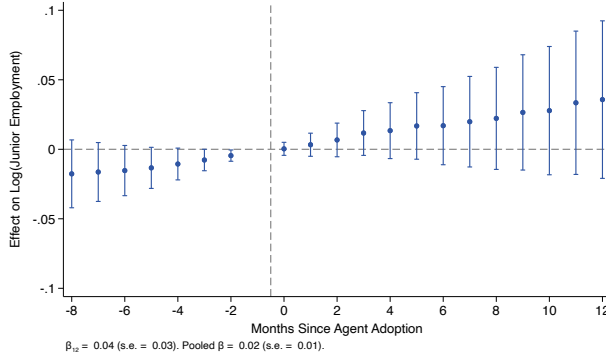
FIGURE 8. Effects of AI on employment



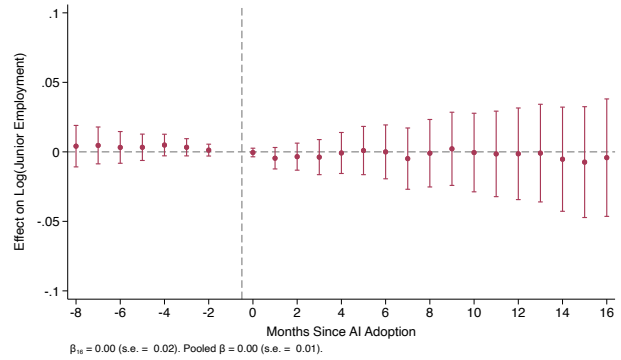
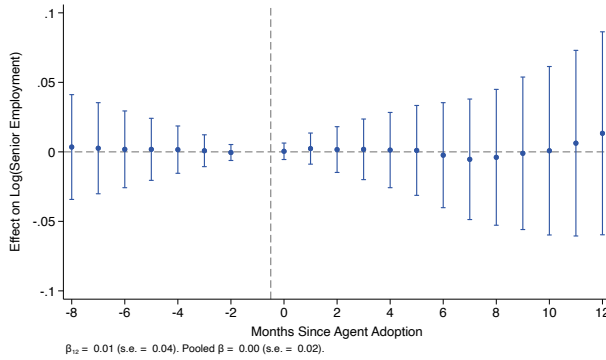
Notes: This figure displays event studies of the effects of AI adoption on log employment. In Panels A and B, the outcome variable is the overall employment, measured as the number of LinkedIn profiles attached to each firm each month (not restricting to engineers). In Panels C and D, the outcome variable is the engineering employment, measured as the number of workers who are active in the JF data in each firm-month. In Panels A and C, the treatment is adoption of AI coding agents. In Panels B and D, 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 9. Effects of AI on employment, by worker seniority

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

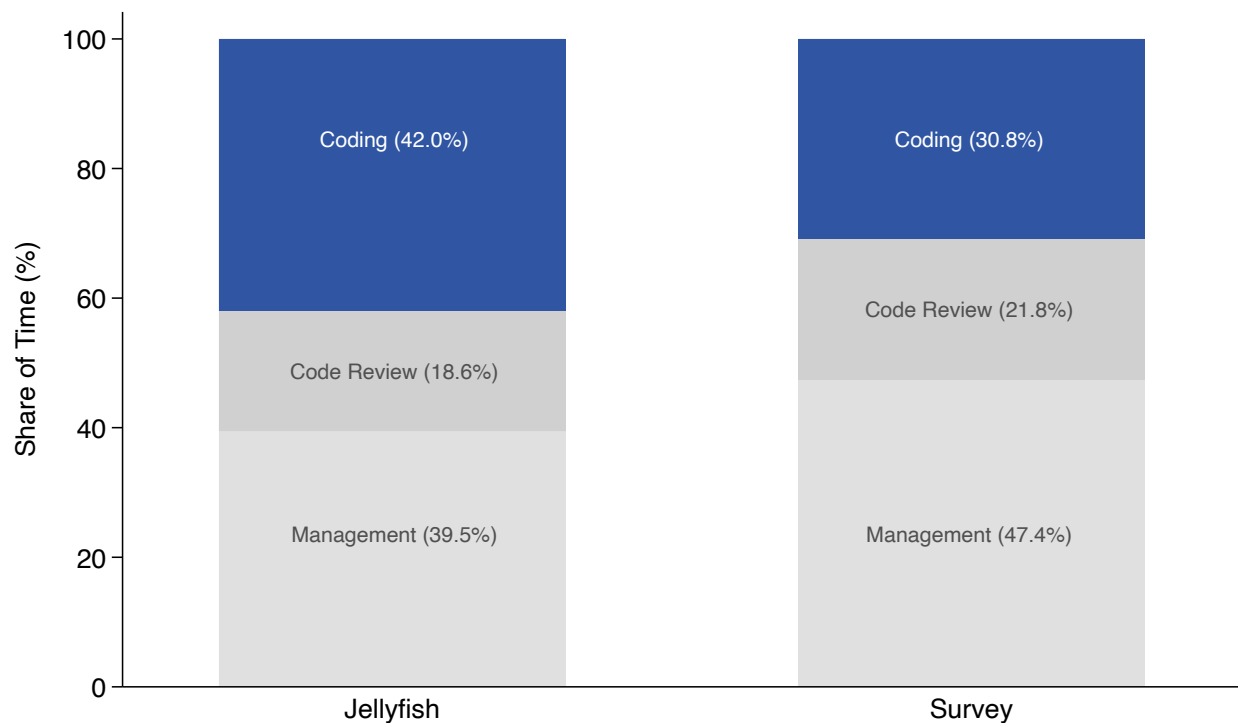


C. AI agents: Overall senior employment (from Revelio) D. AI assistants: Overall senior employment (from Revelio)



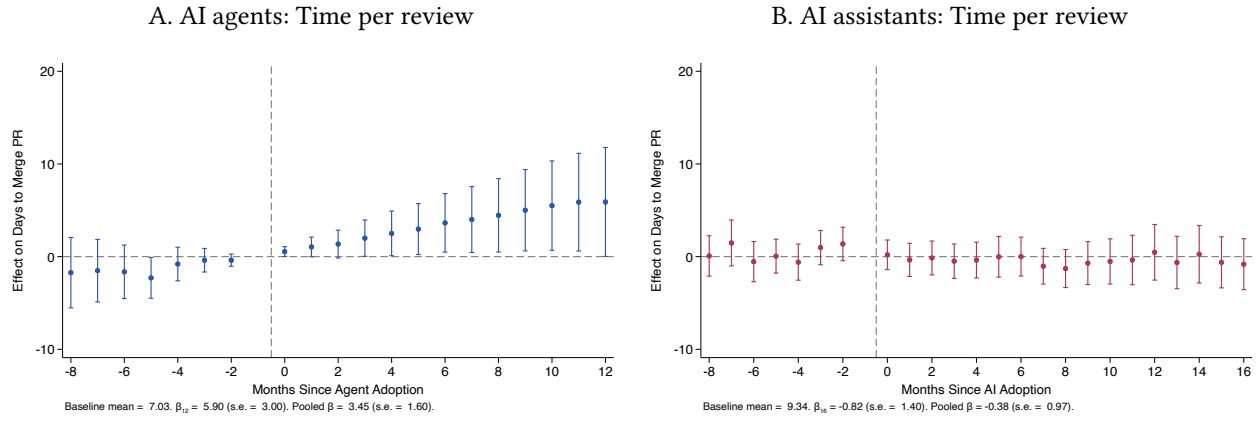
Notes: This figure displays event studies of the effects of AI adoption on log employment, separately by junior and senior workers. In Panels A and B, the outcome variable is the number of LinkedIn profiles of junior workers attached to the firm each month. In Panels C and D, the outcome variable is 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 coding agents. In Panels B and D, 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 10. Coding share of software labor



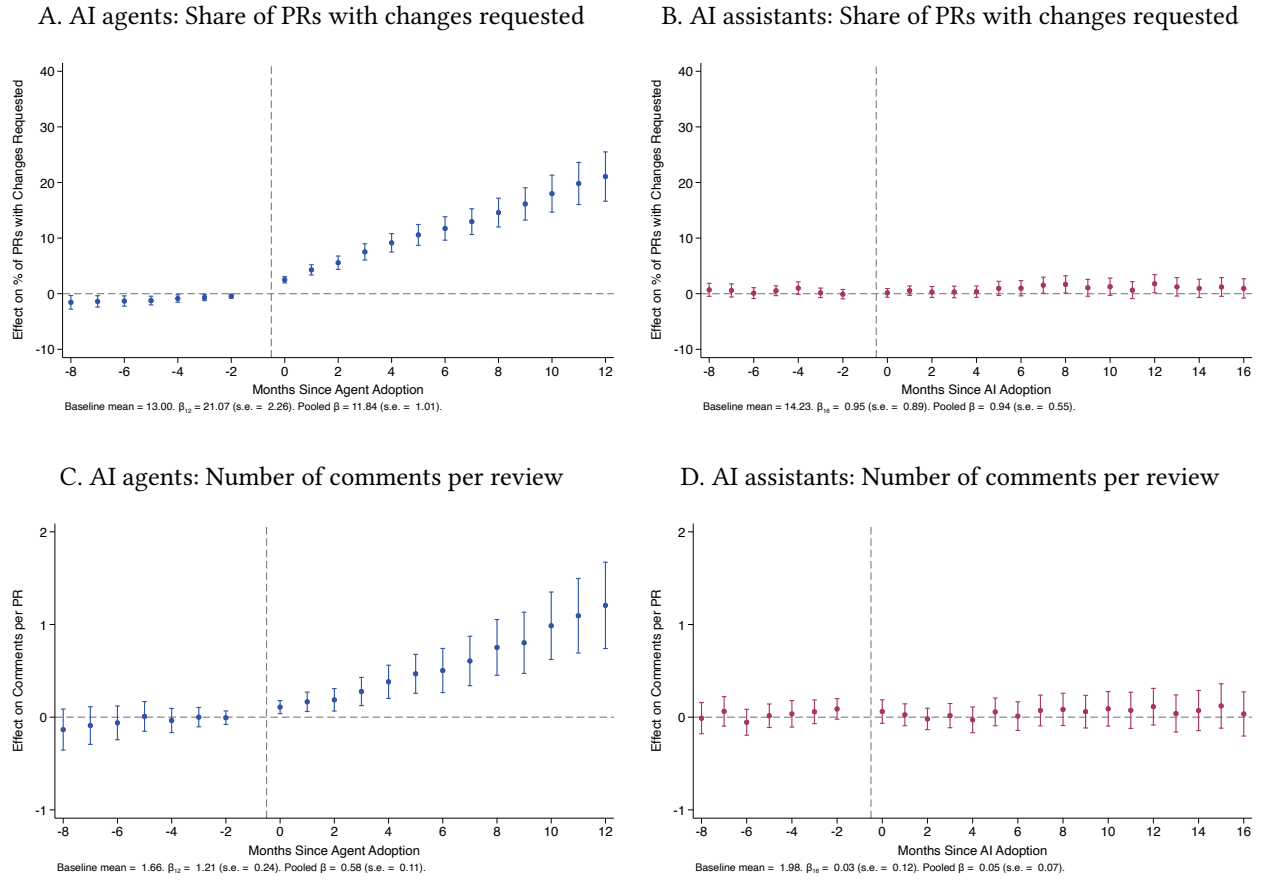
Notes: This figure displays software engineers' average time allocation across three categories of tasks: coding, code review, and management. The first column displays results based on an imputation of time allocation from work signals in the Jellyfish data. We construct a panel of all timestamped work 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.

FIGURE 11. Effects of AI on length of code review



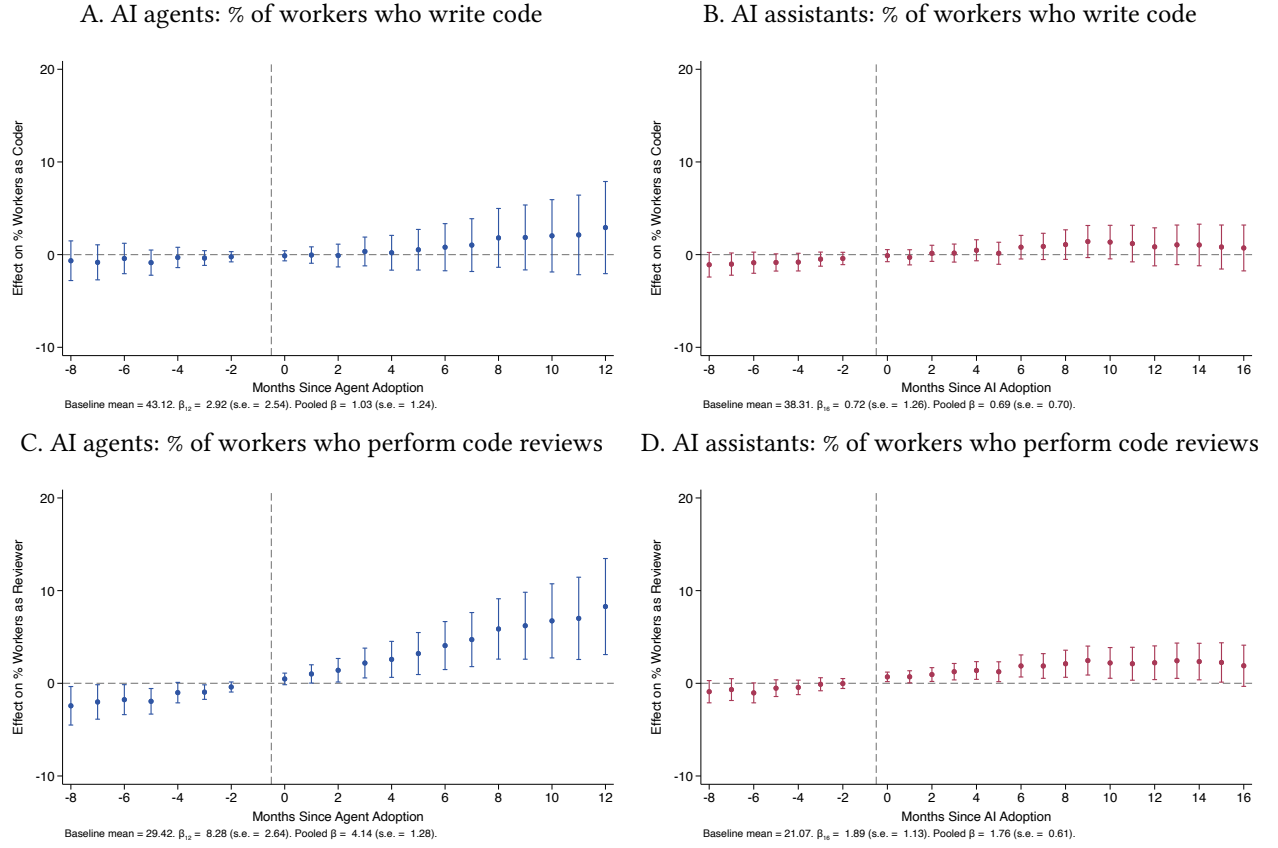
Notes: This figure displays event studies of the effects of AI adoption on the length of the code review process, defined as the number of days between when a pull request is submitted and when it is merged. In Panel A, the treatment is adoption of AI agents. In Panel B, the treatment is adoption of AI assistants. Our specification incorporates time-varying controls for baseline firm size and is implemented using two-way fixed effects.

FIGURE 12. Effects of AI on code review process



Notes: This figure displays event studies of the effects of AI adoption on the code review process. In Panels A and B, the outcome variable is the share of pull requests for which at least one reviewer submitted a formal request for changes. In Panels C and D, the outcome variable is the number of review comments per pull request. In Panels A and C, the treatment is adoption of AI coding agents. In Panels B and D, 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.

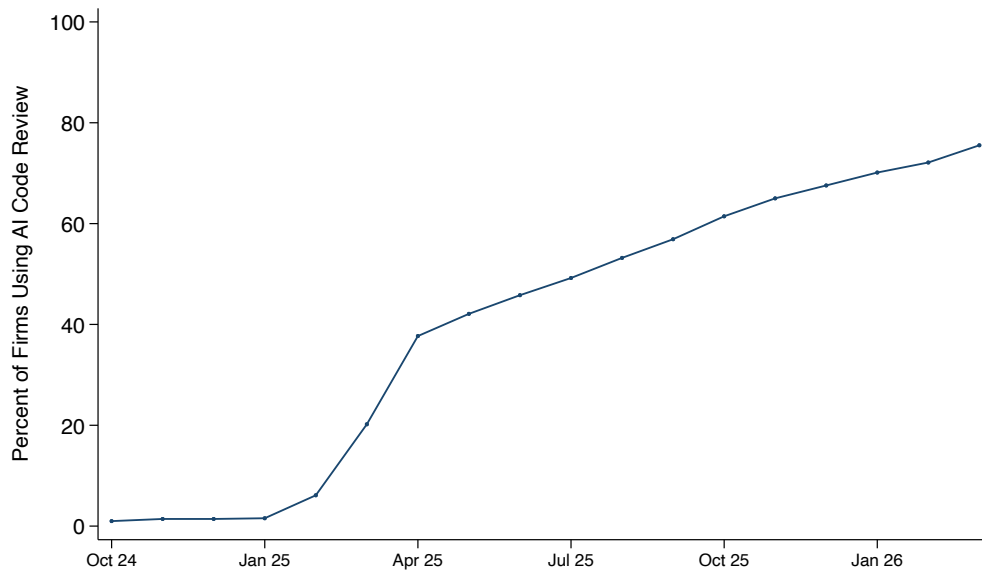
FIGURE 13. Effects of AI on allocation of labor to code writing vs code review



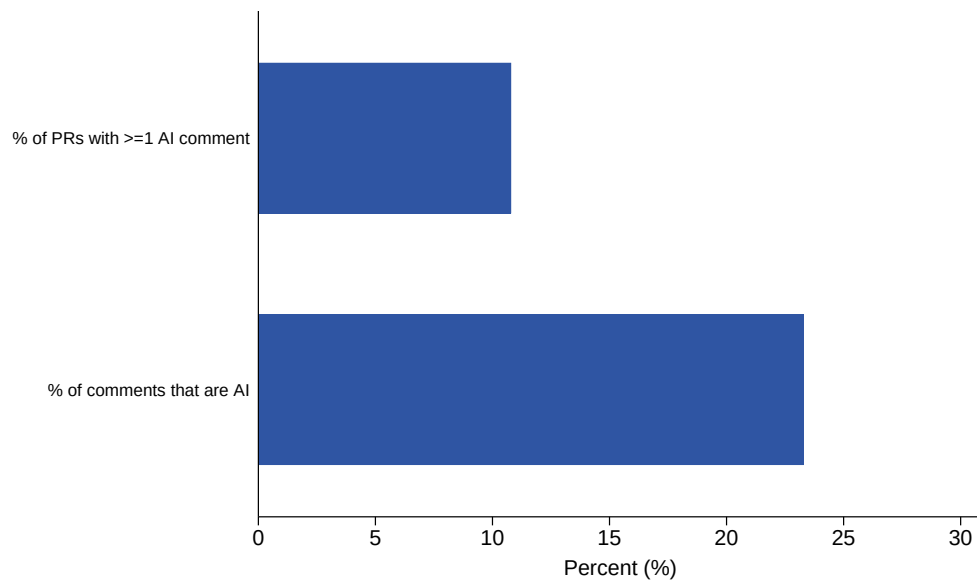
Notes: This figure displays event studies of the effects of AI adoption on the allocation of labor to code writing and code review. In Panels A and B, the outcome variable is the share of workers who make at least one commit in a given firm-month. In Panels C and D, the outcome variable is the share of workers who comment on or approve at least one pull request in a given firm-month. In Panels A and C, the treatment is adoption of AI coding agents. In Panels B and D, 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 14. 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.

B 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.

B.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

B.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

“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

“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

“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

C Tables

TABLE 1. Analysis sample of JF data

A. Size of analysis sample

	Number of firms	Number of workers
Analysis sample	718	725,938

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	99.6	Commits	200,945,700	17.77
		Pull requests	49,050,030	4.34
Task management data	94.4	Issues	68,281,845	6.04
Calendar data	29.7	Events	21,270,247	14.10

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.
<i>Firm size</i>		
Total workers, Jan 2023 (Revelio)	1,652	6,015
Active engineers, Jan 2023 (JF)	241	363
<i>Firm age</i>		
Firm age (years, as of Jan 2023)	19.3	21.4
<i>Revenue engine</i>		
SaaS licence	0.519	
Digital service operator	0.299	
Software as input	0.139	
<i>Customer industry</i>		
Horizontal (business function)	0.245	
Financial services and insurance	0.159	
Healthcare and life sciences	0.116	
Other verticals	0.481	
N	718	

Notes: This table displays summary statistics for the 718 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 licence, digital service operator, or software as input) and customer industry are assigned using an LLM-based classification of firms' Revelio descriptions; NA denotes firms with insufficient description text or no matched Revelio record.

TABLE 3. Worker characteristics

	N	Share (%)
Software engineering	205910	55.2
Data science and research	22813	6.1
Other engineering	2790	0.7
Product and design	31685	8.5
Revenue	55238	14.8
General and administrative	53457	14.3
Chief executives	1019	0.3
Total	372912	100.0

Notes: This table provides an overview of the roles of workers in our data sample. Each worker's role is assigned first from their HR-inputted job title (from Jellyfish) where available, then from their merged LinkedIn job history, where available. Software engineering refers to software, web, mobile, QA, DevOps, infrastructure, and security engineers, together with engineering and information-systems managers. Data science and research covers data scientists, machine-learning and AI engineers, analysts, statisticians, and research scientists. Other engineering captures non-software engineering disciplines—hardware, mechanical, electrical, civil, and industrial engineering. Product and design includes product managers and product owners as well as UX/UI, product, and graphic designers. Revenue comprises sales, marketing, advertising, public relations, market research, and customer success and support. G&A and operations covers back-office and overhead functions—finance and accounting, human resources, legal, operations, administrative support, and general (non-executive) management. Chief executives denotes C-suite officers (e.g., CEO, CFO, CTO), founders, and presidents. Workers for whom job title information is not available are excluded from this table.

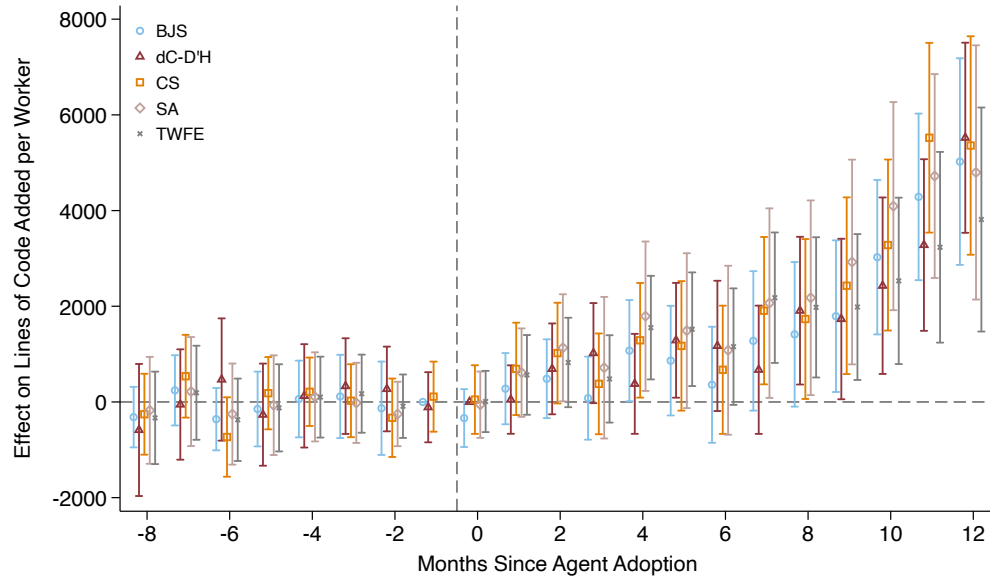
TABLE 4. Balance between early vs late adopters

	AI Assistant Early Adopter (before 2024m1)	AI Agent Early Adopter (before 2025m6)
<i>Firm size</i>		
Log total workers, Jan 2023 (Revelio)	0.047*** (0.013)	0.054*** (0.014)
Log active engineers, Jan 2023 (JF)	0.051*** (0.012)	0.082*** (0.012)
<i>Firm age</i>		
Log firm age (years, as of Jan 2023)	0.019 (0.022)	-0.050** (0.023)
<i>Revenue engine</i>		
SaaS licence	0.055 (0.035)	0.036 (0.037)
Digital service operator	-0.024 (0.038)	0.069* (0.041)
Software as input	-0.061 (0.051)	-0.148*** (0.054)
<i>Customer industry</i>		
Horizontal (business function)	0.048 (0.041)	0.056 (0.043)
Financial services and insurance	-0.104** (0.048)	-0.024 (0.051)
Healthcare and life sciences	0.005 (0.055)	0.019 (0.058)
Other verticals	0.018 (0.035)	-0.036 (0.037)
N	718	718

Notes: This table assesses balance between early and late adopters of AI coding tools. Each entry reports a separate 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; NA denotes firms with insufficient description text or no matched Revelio record. Standard errors in parentheses. *, **, *** indicate significance at the 10%, 5%, and 1% levels.

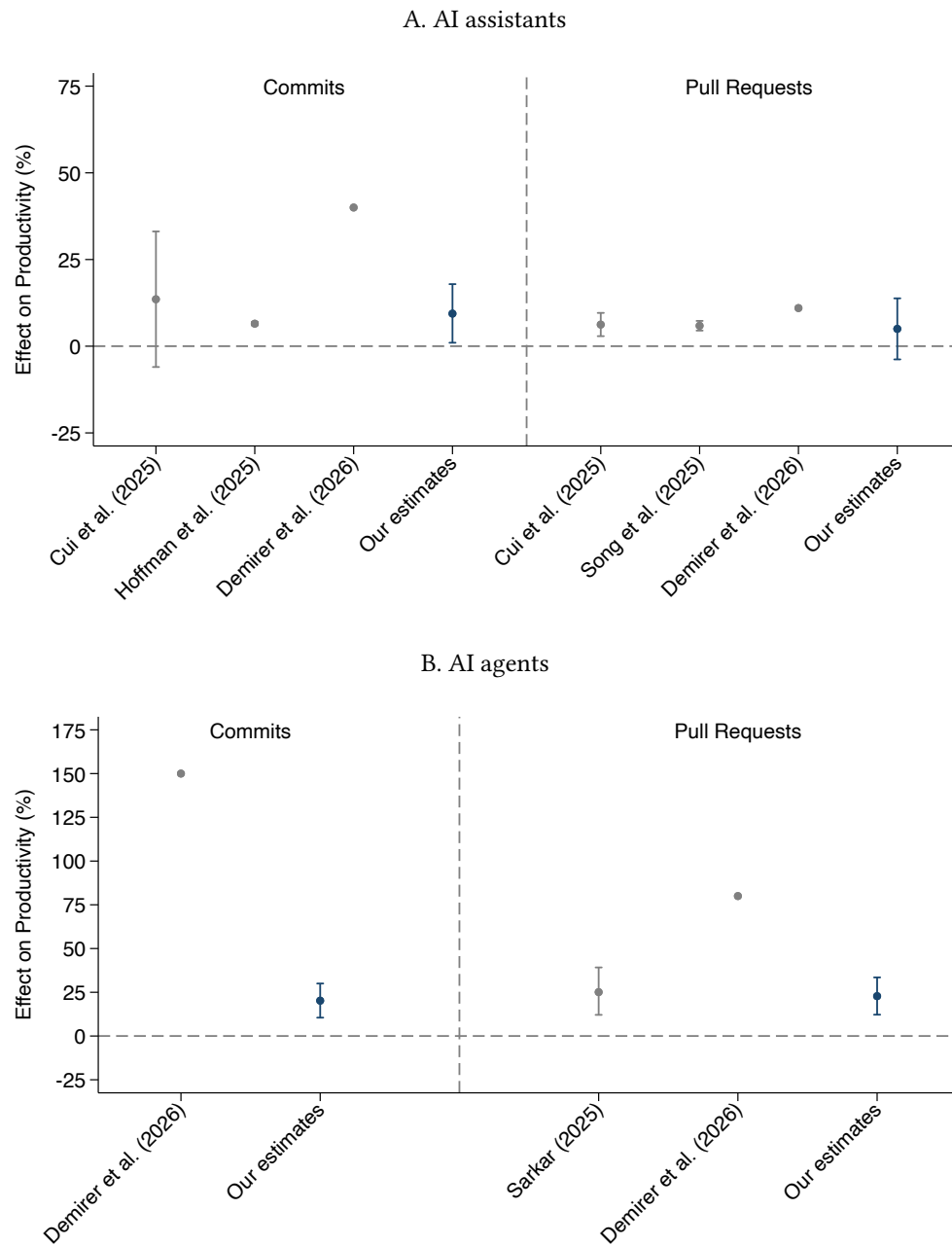
D Appendix Figures

FIGURE 1. Effects of AI on coding activity with different DID estimators



Notes: This figure displays a comparison of estimates using the estimators of Borusyak et al. (2024), de Chaisemartin & D'Haultfœuille (2024), Callaway et al. (2021), and Sun & Abraham (2021), and two-way fixed effects. For illustration, we estimate the effects of agent adoption on lines of code added per worker-month.

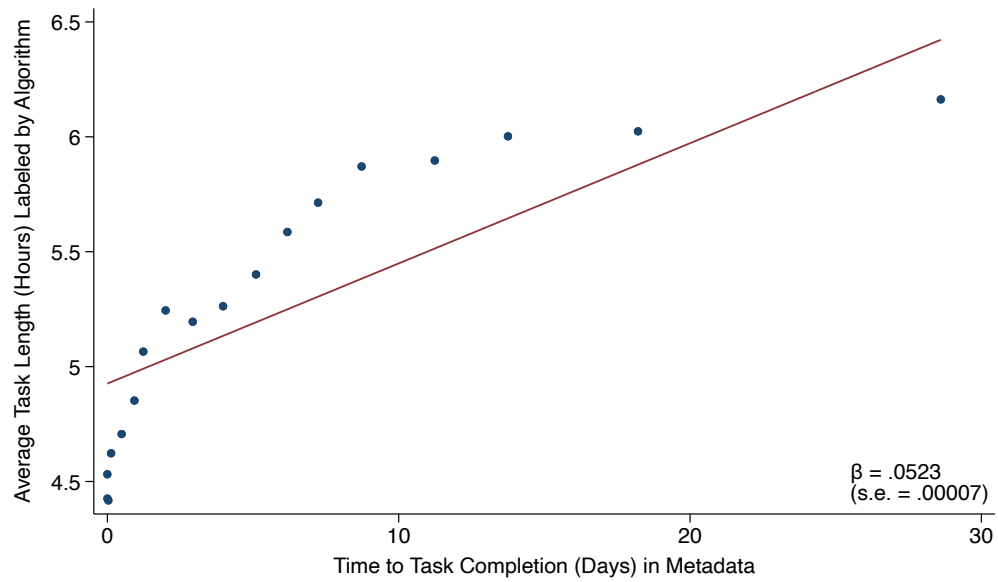
FIGURE 2. Comparison of productivity estimates



Notes: This figure displays a comparison of estimates of AI coding tools' productivity impacts, from our paper with those of Peng et al. (2023); Cui et al. (2024); Hoffmann et al. (2024); Song et al. (2024); Becker et al. (2025); Sarkar (2025), and Demirer, Musolff, & Yang (2026).

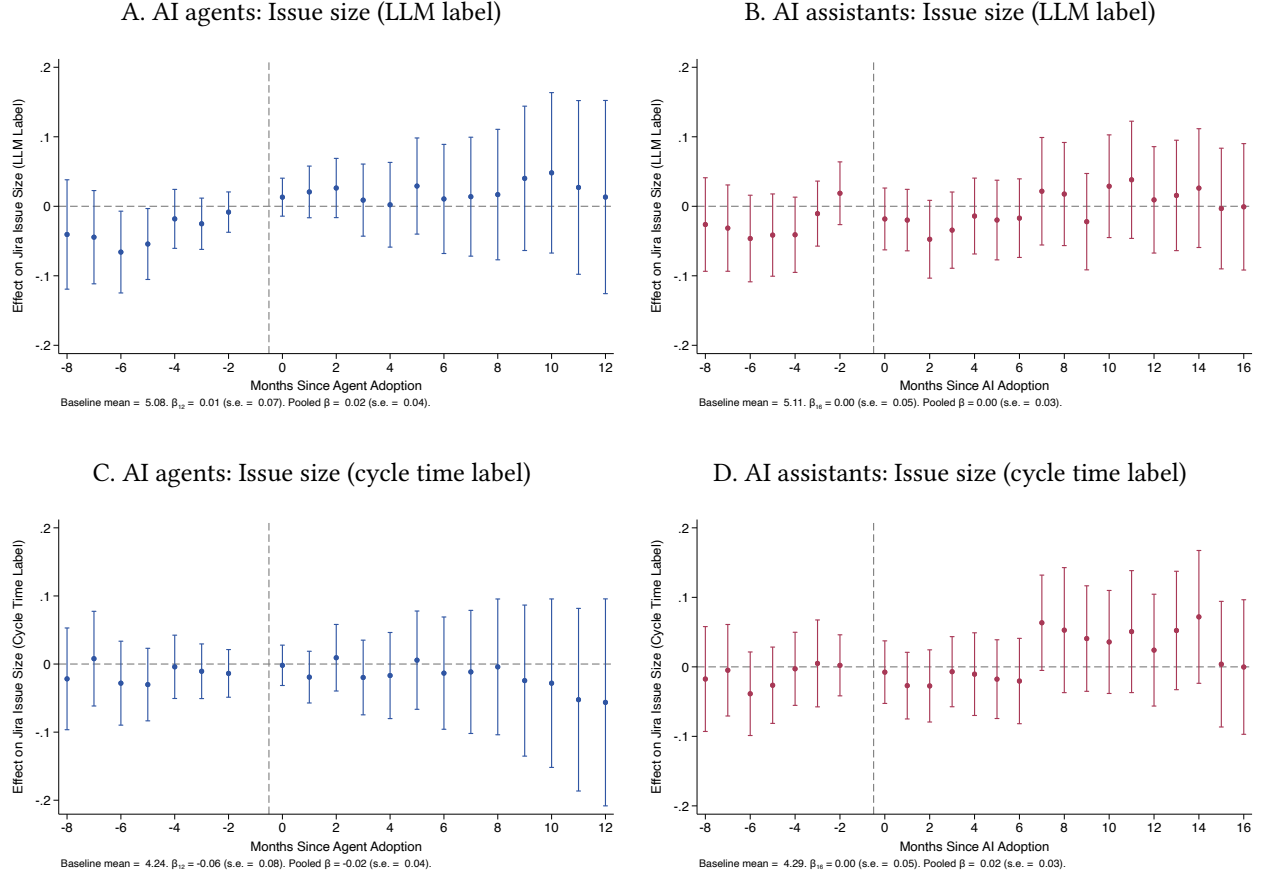
FIGURE 3. Relationship between LLM label and cycle time

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



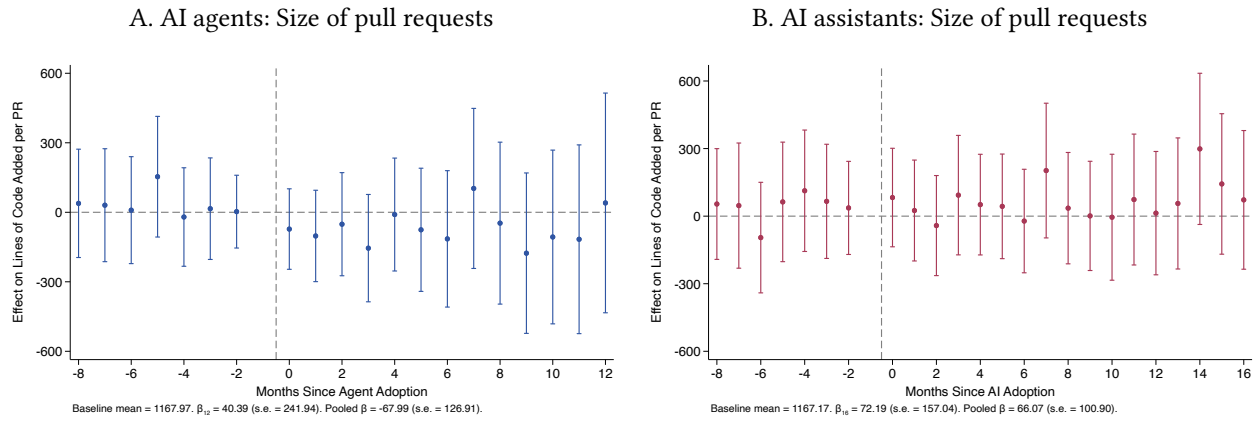
Notes: This figure shows the validation of the “length” label assigned by our machine learning algorithm against days to task completion, as measured in Jira. We winsorize task completion time at 30 days.

FIGURE 4. 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 coding agents. In Panels B and D, 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 5. 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 coding agents. In Panel B, 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.

E Appendix Tables

TABLE 5. Example Jira tasks 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.	Short (<4H)
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.	Long (8-12H)

TABLE 6. 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 a *seat record* for every user who has been assigned a license, including the date on which the seat was activated (`active_date`). Where available, it also records individual query-level usage events (`usage_at`).

Tool codes in the raw data are mapped to three research 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.

Representative AI bot accounts detected in GitHub data (Signal A)

Tool	Login / name pattern
<i>Coding agents</i>	
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
<i>Code review tools</i>	
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
<i>Signal B: commit message patterns</i>	
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
<i>Signal C: PR body patterns</i>	
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 1 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, to represent task descriptions numerically, we extract semantic embeddings using the Sentence-BERT (SBERT) model `all-MiniLM-L6-v2`, a transformer-based architecture designed for sentence-level encoding. This model maps each task’s text into a vector of 384 fixed dimensions.

Second, we train a feedforward neural network in PyTorch to predict the GPT-assigned length label from the SBERT embeddings. The network takes the 384-dimensional embeddings as input, passes them through two hidden layers of sizes 256 and 128, each followed by batch normalization, a ReLU activation, and dropout ($p = 0.3$), and outputs a single continuous value. We train with Smooth L1 (Huber) loss, an Adam optimizer with learning rate 0.001, batch size 512, and 10 training epochs. Evaluated on a held-out 1% test set, the model achieves an MAE of 0.50 on the 0–3 scale.

Finally, we apply the trained model to all tasks, rounding the continuous output to the nearest integer and clipping to $\{0, 1, 2, 3\}$ to obtain discrete length category predictions.

G Proofs Appendix

Proposition 1 (Firm's optimum).

PROOF. First, substitute $L_C = Q/A$ and $L_R = rQ$ into the firm's problem:

$$\max_{Q,r} R(Q(1-\pi)) - Q \left[\frac{w_C}{A} + w_R r + \ell\pi(1-d(r)) \right].$$

Take the first-order condition with respect to r :

$$-Q [w_R - \ell\pi d'(r^*)] = 0,$$

which implies

$$d'(r^*)\ell\pi = w_R.$$

Similarly, take the first-order condition with respect to Q :

$$(1-\pi)R'(Q^*(1-\pi)) = \frac{w_C}{A} + w_R r^* + \ell\pi[1-d(r^*)] \equiv c^*.$$

where c^* denotes the total cost of producing an extra unit of code, including the cost of reviewers and the expected losses from bugs. Finally, the labor constraints imply:

$$L_C^* = \frac{Q^*}{A}, \quad L_R^* = r^* Q^*.$$

□

Proposition 2 (Effects of increase in coding productivity).

PROOF. First, we obtain an expression for $\frac{d \ln r^*}{d \ln A}$. The review FOC is

$$d'(r^*)\ell\pi = w_R,$$

which does not depend on A . Hence

$$\frac{d \ln r^*}{d \ln A} = 0.$$

Next, we obtain an expression for $\frac{d \ln Q^*}{d \ln A} = \frac{d \ln G^*}{d \ln A}$. The output FOC is

$$(1-\pi)R'(G^*) = c^*.$$

Taking logs gives

$$\ln(1-\pi) + \ln R'(G^*) = \ln c^*.$$

Holding π fixed and differentiating with respect to $\ln A$,

$$\frac{d \ln R'(G^*)}{d \ln G^*} \frac{d \ln G^*}{d \ln A} = \frac{d \ln c^*}{d \ln A}.$$

Recall that, by definition,

$$-\frac{d \ln R'(G^*)}{d \ln G^*} := \rho$$

and, by the envelope theorem,

$$\frac{d \ln c^*}{d \ln A} = -\frac{w_C/A}{c^*} = -s_C.$$

Substituting these two expressions in, we get

$$\begin{aligned} -\rho \frac{d \ln G^*}{d \ln A} &= -s_C \\ \frac{d \ln G^*}{d \ln A} &= \frac{s_C}{\rho}. \end{aligned}$$

Because π is fixed and $G^* = Q^*(1 - \pi)$,

$$\frac{d \ln Q^*}{d \ln A} = \frac{d \ln G^*}{d \ln A} = \frac{s_C}{\rho}.$$

Finally, we obtain expressions for $\frac{d \ln L_C^*}{d \ln A}$ and $\frac{d \ln L_R^*}{d \ln A}$.

$$\begin{aligned} L_C^* = \frac{Q^*}{A} &\Rightarrow \frac{d \ln L_C^*}{d \ln A} = \frac{d \ln Q^*}{d \ln A} - 1 = \frac{s_C}{\rho} - 1, \\ L_R^* = r^* Q^* &\Rightarrow \frac{d \ln L_R^*}{d \ln A} = \frac{d \ln r^*}{d \ln A} + \frac{d \ln Q^*}{d \ln A} = \frac{s_C}{\rho}. \end{aligned}$$

□

Proposition 3 (Bottleneck from productivity channel).

PROOF. Suppose $L_R = \bar{L}_R$ is fixed, so that $r = \bar{L}_R/Q$. Substituting into the firm's problem, and noting that the term $w_R \bar{L}_R$ is a constant that does not affect the choice of Q , the firm solves

$$\max_Q R(Q(1 - \pi)) - \frac{w_C}{A} Q - \ell Q \pi \left[1 - d\left(\frac{\bar{L}_R}{Q}\right) \right].$$

The first-order condition is

$$(1 - \pi)R'(Q(1 - \pi)) = \frac{w_C}{A} + \ell \pi [1 - d(r)] + \ell \pi d'(r)r \equiv \tilde{c}(Q), \quad r = \frac{\bar{L}_R}{Q}.$$

Evaluate this at a point where $\bar{L}_R = r^* Q$, so that $r = r^*$; using the review first-order condition $w_R = \ell \pi d'(r^*)$ from Proposition 1, $\tilde{c}(Q) = c^*$, so the constrained and unconstrained marginal costs of code coincide at this point.

Differentiating $\ln(1 - \pi) + \ln R'(Q(1 - \pi)) = \ln \tilde{c}(Q)$ with respect to $\ln Q$ and $\ln A$, holding π fixed, and

using $-\frac{d \ln R'(G^*)}{d \ln G^*} = \rho$,

$$-\rho d \ln Q = \frac{\partial \ln \tilde{c}}{\partial \ln A} d \ln A + \frac{\partial \ln \tilde{c}}{\partial \ln Q} d \ln Q.$$

Only the first term of $\tilde{c}(Q)$ depends on A , so $\frac{\partial \ln \tilde{c}}{\partial \ln A} = -\frac{w_C/A}{c^*} = -s_C$, exactly as in Proposition 2. For the second term, write the r -dependent part of $\tilde{c}$ as $h(r) := \ell\pi[1 - d(r)] + \ell\pi d'(r)r$, so that

$$h'(r) = -\ell\pi d'(r) + \ell\pi d''(r)r + \ell\pi d'(r) = \ell\pi d''(r)r.$$

Since $r = \bar{L}_R/Q$, we have $\frac{dr}{d \ln Q} = -r$. Hence, evaluated at $r = r^*$,

$$\left. \frac{\partial \ln \tilde{c}}{\partial \ln Q} \right|_{r=r^*} = \frac{1}{c^*} h'(r^*) \frac{dr}{d \ln Q} = \frac{1}{c^*} [\ell\pi d''(r^*) r^*] (-r^*) = -\frac{\ell\pi d''(r^*) (r^*)^2}{c^*}.$$

Using $\lambda \equiv -\frac{r^* d''(r^*)}{d'(r^*)}$ and $s_R \equiv \frac{w_R r^*}{c^*} = \frac{\ell\pi d'(r^*) r^*}{c^*}$, this becomes

$$\left. \frac{\partial \ln \tilde{c}}{\partial \ln Q} \right|_{r=r^*} = \lambda s_R.$$

Substituting into the differentiated first-order condition,

$$-\rho d \ln Q = -s_C d \ln A + \lambda s_R d \ln Q \implies \left. \frac{d \ln Q^*}{d \ln A} \right|_{\bar{L}_R} = \frac{s_C}{\rho + \lambda s_R}.$$

Since $\lambda, s_R > 0$, this is strictly smaller than s_C/ρ , the pass-through under flexible reallocation established in Proposition 2. Because π is fixed, $G^* = Q^*(1 - \pi)$, so the same expression holds for $\left. \frac{d \ln G^*}{d \ln A} \right|_{\bar{L}_R}$. $\square$

Proposition 4 (Effects of increase in the bug rate, bottleneck from bug rate channel).

PROOF. First, we obtain an expression for $\frac{d \ln r^*}{d \ln \pi}$. The review FOC is

$$d'(r^*) \ell \pi = w_R.$$

Taking logs gives

$$\ln d'(r^*) + \ln \ell + \ln \pi = \ln w_R.$$

Differentiating with respect to $\ln \pi$,

$$\frac{d \ln d'(r^*)}{d \ln r^*} \frac{d \ln r^*}{d \ln \pi} + 1 = 0.$$

Recall that, by definition,

$$\lambda \equiv -\frac{d \ln d'(r^*)}{d \ln r^*},$$

we obtain

$$\frac{d \ln r^*}{d \ln \pi} = \frac{1}{\lambda}.$$

Next, we obtain expressions for $\frac{d \ln G^*}{d \ln \pi}$ and $\frac{d \ln Q^*}{d \ln \pi}$. The output FOC is

$$(1 - \pi)R'(G^*) = c^*.$$

Taking logs,

$$\ln(1 - \pi) + \ln R'(G^*) = \ln c^*.$$

Differentiating with respect to $\ln \pi$ gives

$$-\frac{\pi}{1 - \pi} + \frac{d \ln R'(G^*)}{d \ln G^*} \frac{d \ln G^*}{d \ln \pi} = \frac{d \ln c^*}{d \ln \pi}.$$

By the envelope theorem, since r^* is chosen optimally,

$$\frac{dc^*}{d\pi} = \ell[1 - d(r^*)],$$

so

$$\frac{d \ln c^*}{d \ln \pi} = \frac{\ell\pi[1 - d(r^*)]}{c^*} = s_B.$$

Substituting this and

$$\frac{d \ln R'(G^*)}{d \ln G^*} = -\rho$$

yields

$$-\frac{\pi}{1 - \pi} - \rho \frac{d \ln G^*}{d \ln \pi} = s_B,$$

or

$$\frac{d \ln G^*}{d \ln \pi} = -\frac{1}{\rho} \left(s_B + \frac{\pi}{1 - \pi} \right).$$

Since $G^* = Q^*(1 - \pi)$,

$$\frac{d \ln G^*}{d \ln \pi} = \frac{d \ln Q^*}{d \ln \pi} - \frac{\pi}{1 - \pi}.$$

Therefore,

$$\frac{d \ln Q^*}{d \ln \pi} = -\frac{1}{\rho} \left(s_B + \frac{\pi}{1 - \pi} \right) + \frac{\pi}{1 - \pi}.$$

Finally, we obtain expressions for $\frac{d \ln L_C^*}{d \ln \pi}$ and $\frac{d \ln L_R^*}{d \ln \pi}$. Because $L_C^* = Q^*/A$ and A is held fixed,

$$\frac{d \ln L_C^*}{d \ln \pi} = \frac{d \ln Q^*}{d \ln \pi}.$$

Finally, since $L_R^* = r^* Q^*$,

$$\frac{d \ln L_R^*}{d \ln \pi} = \frac{d \ln r^*}{d \ln \pi} + \frac{d \ln Q^*}{d \ln \pi} = \frac{1}{\lambda} + \frac{d \ln Q^*}{d \ln \pi}.$$

□

Corollary 1 (Overall effects of AI).

PROOF. The result follows by applying the chain rule to Propositions 2 and 4. For any outcome $Y \in \{r^*, G^*, Q^*, L_C^*, L_R^*\}$,

$$\frac{\ln Y}{\ln t} = \frac{\ln Y \ln A}{\ln A \ln t} + \frac{\ln Y \ln \pi}{\ln \pi \ln t}.$$

Substituting the comparative statics from Propositions 2 and 4 gives the expressions in the corollary. For item 2, for instance, Proposition 2 gives $\frac{d \ln G^*}{d \ln A} = \frac{s_C}{\rho}$ and Proposition 4 gives $\frac{d \ln G^*}{d \ln \pi} = -\frac{1}{\rho} \left(s_B + \frac{\pi}{1-\pi} \right)$, so

$$\frac{d \ln G^*}{d \ln t} = \frac{s_C}{\rho} \frac{d \ln A}{d \ln t} - \frac{1}{\rho} \left(s_B + \frac{\pi}{1-\pi} \right) \frac{d \ln \pi}{d \ln t},$$

which is the stated expression. The remaining items follow the same way. $\square$

Proposition 5 (Employment effects under limited output pass-through).

PROOF. The result follows from Corollary 1. As $\rho \rightarrow \infty$, $1/\rho \rightarrow 0$. Thus, we can simplify the expressions for the effects on coding and review labor:

$$\frac{d \ln L_C^*}{d \ln t} = -\frac{d \ln A}{d \ln t} + \frac{\pi}{1-\pi} \frac{d \ln \pi}{d \ln t}$$

and

$$\frac{d \ln L_R^*}{d \ln t} = \left(\frac{\pi}{1-\pi} + \frac{1}{\lambda} \right) \frac{d \ln \pi}{d \ln t}$$

$\square$

Corollary 2 (Relative allocation of labor between coding and review).

PROOF. From Proposition 1, $L_C^* = Q^*/A$ and $L_R^* = r^* Q^*$, so

$$\frac{L_R^*}{L_C^*} = \frac{r^* Q^*}{Q^*/A} = r^* A.$$

Taking logs, $\ln(L_R^*/L_C^*) = \ln r^* + \ln A$. Differentiating with respect to $\ln A$, holding π fixed, and using $\frac{d \ln r^*}{d \ln A} = 0$ from Proposition 2,

$$\frac{d \ln(L_R^*/L_C^*)}{d \ln A} = \frac{d \ln r^*}{d \ln A} + 1 = 1.$$

Differentiating with respect to $\ln \pi$, holding A fixed, and using $\frac{d \ln r^*}{d \ln \pi} = \frac{1}{\lambda}$ from Proposition 4,

$$\frac{d \ln(L_R^*/L_C^*)}{d \ln \pi} = \frac{d \ln r^*}{d \ln \pi} = \frac{1}{\lambda}.$$

By the chain rule,

$$\frac{d \ln(L_R^*/L_C^*)}{d \ln t} = \frac{d \ln(L_R^*/L_C^*)}{d \ln A} \frac{d \ln A}{d \ln t} + \frac{d \ln(L_R^*/L_C^*)}{d \ln \pi} \frac{d \ln \pi}{d \ln t} = \frac{d \ln A}{d \ln t} + \frac{1}{\lambda} \frac{d \ln \pi}{d \ln t},$$

which is the stated expression. $\square$

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
- Amodei, D. (2024, October). *Machines of Loving Grace*. Retrieved 2026-06-29, from <https://darioamodei.com/essay/machines-of-loving-grace>
- 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. (2022, May). *The Labor Market Impacts of Technological Change: From Unbridled Enthusiasm to Qualified Optimism to Vast Uncertainty* [Working Paper]. National Bureau of Economic Research. Retrieved 2024-10-09, from <https://www.nber.org/papers/w30074> doi: 10.3386/w30074
- 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.
- 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
- 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., Goodman-Bacon, A., & Sant’Anna, P. H. (2021). Difference-in-differences with a continuous treatment. *arXiv preprint arXiv:2107.02637*.
- 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
- 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
- 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
- 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.
- Eloundou, T., Manning, S., Mishkin, P., & Rock, D. (2024). Gpts are gpts: Labor market impact potential of llms. *Science*, 384(6702), 1306–1308.
- 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>
- 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
- 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
- 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
- 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.
- 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/>
- 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
- 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
- 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
- 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).
- Manzoor, E., Ascarza, E., & Netzer, O. (2025, November). *Learning When to Quit in Sales Conversations*.
- 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
- Ng, A. (2017, February). *Andrew Ng: Artificial Intelligence is the New Electricity*. Retrieved 2026-06-29, from <https://www.youtube.com/watch?v=21EiKfQYZXc>
- 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 ed.). Dubuque, IA: McGraw-Hill.
- 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)
- 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=getftr&utm_medium=getftr&utm_campaign=getftr_pilot&utm_integrator=wiley
- 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*.
- Stackpole, B. (2024, August). *The impact of generative AI as a general-purpose technology*. Retrieved 2026-06-29, from <https://mitsloan.mit.edu/ideas-made-to-matter/impact-generative-ai-a-general-purpose-technology>
- 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
- VandeHei, J., & Allen, M. (2025, May). *AI jobs danger: Sleepwalking into a white-collar bloodbath*. Retrieved 2026-06-29, from <https://www.axios.com/2025/05/28/ai-jobs-white-collar-unemployment-anthropic>
- 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*.