AI Agents, Productivity, and Higher-Order Thinking: Early Evidence From Software Development

Suproteem K. Sarkar
University of Chicago

November 6, 2025 (Preprint)

Link to latest version

How are agentic AI systems affecting productivity and the cognitive experience of work? This paper studies the effects of agents on software development by analyzing the impacts of an AI-assisted programming platform. In 2024, the platform released a coding agent that allows its users to generate code from natural language instructions. While the majority of users work in software roles, there is also a cohort of frequent agent users across non-engineering roles like design and product management. More experienced workers are more likely to accept agent-generated code—one standard deviation higher work experience corresponds to 6% higher accept rates. This positive experience gradient for agent accepts contrasts with a negative experience gradient for AI autocompletion accepts, as well as a junior worker skew among other kinds of AI tools. A difference-in-differences analysis of weekly code merges indicates that software output increased by 39% after the agent became the platform's default code generation mode. Agents may shift the production process from the syntactic activity of typing code to the semantic activity of instructing and evaluating agents. User messages to agents include instructions to implement, explain, and plan. Experienced workers are more likely to develop plans in their initial messages to agents, which could improve alignment with user intent. These results suggest that abstraction, clarity, and evaluation may be important skills for workers.

I thank Marianne Bertrand, Jiafeng Chen, Joshua D. Gottlieb, Anders Humlum, Emir Kamenica, Dennis Lee, Jacob D. Leshno, Yueran Ma, Luke Melas-Kyriazi, Amil Merchant, Max Miller, Sendhil Mullainathan, Karl Oskar Schulz, Joshua Schwartzstein, Rohan Shah, Andrei Shleifer, Adi Sunderam, Mirac Suzgun, and seminar participants at the University of Chicago and Yale. Part of this research was conducted while I was a visiting researcher at Anysphere in the summer of 2025. Some companies that shared data for this study had a limited right to review the manuscript prior to its release, in order to identify specific instances of confidential or personally identifiable information. I had editorial discretion over the paper and the right to publish. Sarkar: University of Chicago, Booth School of Business (suproteem@uchicago.edu)

An agentic AI system pursues objectives by perceiving and acting on its environment (Russell and Norvig, 2020; Wooldridge and Jennings, 1995). Recent advances in agentic AI systems integrate language models with external tools and feedback loops (Yao et al., 2023; Schick et al., 2023). With these integrations, an agent can access information and take actions that may be unavailable to a standalone language model. While many applications of AI have provided workers with recommendations and guidance (Friedman, 2021; Angelova et al., 2023), agents allow workers to delegate some self-contained tasks to AI systems.

This paper finds evidence consistent with four economic implications of AI agents applied to production. First, agents allow non-specialists to perform some specialist tasks. Second, workers with more experience and organizational context appear to have higher skill delegating tasks to agents. Third, agent usage is associated with increases in organization-level output. Fourth, agents facilitate a shift from syntactic to semantic work patterns—from implementing tasks to specifying objectives and evaluating responses.

Software development has been one of the earliest professions to adopt AI agents. This paper analyzes adoption, usage, messages, and output related to Cursor Agent—a coding agent that converts natural language instructions into generated code. While agents are primarily used by software engineers, they are also used by people in non-engineering roles like design and product management. Experienced workers accept agent-generated code more often—this contrasts with other kinds of AI tools that have been disproportionately applied to output by less experienced workers.

Agent usage is associated with higher output. Organization-level data on weekly code merges indicates that software output increased by 39% after the agent was made the default generation mode for eligible users. There were no significant changes in merge revert rates in the six months following the agent's release—this result is consistent with similar rates of production issues from code changes, but does not measure longer-run impacts on codebase structure and maintainability. Cross-sectionally, frequent agent users tend to produce more output and implement larger-scoped code changes.

Agents may influence the cognitive experience of production. Software production has traditionally involved typing code sequences into a computer. Agent users produce software by providing natural language instructions and evaluating generated code sequences. Usage is multipurpose—people use agents not just to implement code, but also to explain code behavior and develop plans. More experienced workers are more likely to generate plans in their initial messages to agents, which may improve alignment between agent trajectories and user intent.

Adoption of agents To measure how AI has been adopted for software development and how people use agents, I study a sample of 1,000 organizations that adopted Cursor—a platform for AI-assisted programming. The platform is used by people working in sectors including software, finance, and media. It is primarily used by people working in software roles, who make up 76% of users in the sample. It is also used by people working in other roles like data science, product management, and design. The platform is used by both individual contributors and managers, and is adopted across levels of work experience.

In November 2024, the platform released a coding agent that integrates language models with external tools. These tools include codebase search, web browsing, and command execution, and allow agents to interact with codebases and external data. The agent feature is adopted by most users of the platform—by July 2025, the feature was used by 92% of active users and all active organizations in the sample.

While most agent users work in software roles, agents are also used by people working in roles and sectors with traditionally lower engineering intensity—including managers, designers, and workers at consulting firms. Software engineers submit 52.1 agent messages per week. Meanwhile, designers submit 74.7 agent messages per week and accept code 1.14 times as often as software engineers do. Given current agent capabilities, people may be more likely to delegate tasks that are verifiable over short horizons to agents.

Experience gap in agent accept rates Among agent users, the most frequent usage is at the top and bottom deciles of work experience. Experienced workers are more likely to accept agent-generated code. This difference in accept rates may be a product of experienced workers' higher skill using agents, task distribution better suited for the feature, or more liberal accept criteria.

Additional evidence is consistent with a skill-based explanation for this experience gap in agent accept rates. The experience gap persists after adjusting for role, sector, and seniority—including manager status. If the experience gap were fully explained by task distributions better suited for agents, there would need to be commensurate variation in experienced workers' output tasks not accounted for by these observable characteristics. Experienced workers are also more likely to develop plans with agents before acting, which suggests that generated output may be more aligned with these workers' intent. If higher accept rates were driven by less vigilant agent use, it would be unlikely for experienced workers to have plan-first usage styles. Moreover, experienced

workers are less likely to request that agents explain code, which suggests they may have more codebase context that could improve alignment.

The experience gap in accept rates differs from usage patterns of other kinds of AI tools, which have been skewed toward less experienced workers. For example, suggestions from the Cursor autocompletion feature, an earlier AI-assisted programming technology, are more likely to be accepted by people with less work experience. Prior research that analyzes ChatGPT usage has also found a junior skew: Humlum and Vestergaard (2025) find that the platform was more likely to be used by younger workers; Chatterji et al. (2025) find that younger people are a higher share of users, though older people are more likely to send work-related messages. One potential explanation is that AI autocompletion tools assist during task execution, while AI agents require task specification. People with more experience and organizational context may have higher skill specifying the tasks that they delegate.

Changes in output after agent release To quantify changes in outcomes associated with the agent feature's release, I analyze merge characteristics from a sample of 32 organizations. The agent feature was released on November 24, 2024 and was made the default generation mode on February 19, 2025. I use a difference-in-differences design that compares outcomes in the weeks surrounding the agent's release. The analysis period covers merges from October 2024 to May 2025. The comparison is between 24 organizations that were already using the platform when the analysis period began (the "eligible group") and 8 organizations that were not using the platform during the analysis period (the "baseline group").

Following the agent feature's release, software output among eligible organizations increased. In the 28-week period after the agent was released, weekly code merges were 26% higher in the eligible group relative to time trends in the baseline group. Filtering to the 15-week period after the agent was made the default generation mode, weekly code merges were 39% higher in the eligible group relative to time trends in the baseline group. There were no significant changes in revert rates and decreases in bugfix rates after the feature release—these results are consistent with similar codebase quality in the short run, but do not measure structural, longer-horizon changes in quality from agent use. There were no significant changes in lines edited per merge or files touched per merge, which is consistent with similar scope of additional output after the agent's release.

Using cross-sectional data from one organization, I find that more frequent agent use

is associated with higher merge output, as well as more lines edited and files touched per merge. Unlike the agent release results, these cross-sectional results are not the product of a natural experiment: While more frequent agent use may increase output or scope, more productive workers could also be more disposed to use agents frequently.

Early adopters of the platform may not be representative of all organizations that develop software. If early adopters are more willing to use new technologies and have greater skill using agents, these estimated effects may be larger than effects of coding agent use on other organizations. On the other hand, productivity effects of information technologies can increase with time as complements mature (Brynjolfsson et al., 2021)—the effects of agents on output may also increase as technologies and adoption patterns evolve.

Semantic shift in work patterns People who use agents produce software using a natural language interface—this usage style may influence the experience of programming. Traditionally, developers have translated high-level goals into software by manually typing code sequences. Using agents, developers can specify goals using natural language, and then evaluate and deploy AI-generated code sequences. If agents help people map semantics to syntax, work patterns may shift toward developing and specifying semantics, and evaluating agent-produced syntax.

To study work patterns with agents, I analyze user message intent. I study the characteristics of first messages to agents across a sample of 1,000 users. While the majority (61%) of messages are for implementation, people also use agents to explain code and to make plans. Explanation messages may help people understand codebases and code behavior, and refine the semantics of the software they aim to produce. Planning messages may help people turn goals into implementation logic, and may also give people opportunities to modify an agent's behavior before it generates code. Users with plan-first usage styles have higher agent accept rates.

Experience, planning, and context People with more work experience are more likely to instruct agents to plan before acting. This behavior may improve alignment—it gives users an opportunity to review and modify agent behavior before changes are made. People with more work experience are also less likely to instruct agents to explain code. Put together, these results suggest that experienced workers are more likely to seek alignment before generating output, and may have more codebase understanding.

Effective agent use may involve fluency with abstraction, clarity, and evaluation. An

agent user has an implicit model of what agents are capable of producing. Alignment of agent-produced output may depend on how clearly a user specifies their intent. Alignment between output and intent may also depend on how effectively a user evaluates generated code before applying it to output.

Related literature This paper builds on the literature studying economics, technology, and cognition. The impacts of information technologies vary across task types, management practices, and organization characteristics, and may not be fully realized in the short run (Mansfield, 1961; Bresnahan and Trajtenberg, 1995; Brynjolfsson and Hitt, 2000; Athey and Stern, 2002; Autor et al., 2003; Acemoglu and Autor, 2011; Syverson, 2011; Bloom et al., 2014; Acemoglu et al., 2024). Computation can scaffold memory and reasoning, and can offload mental processes to external environments (Hutchins, 2006; Kirsh and Maglio, 1994; Zhang and Norman, 1994; Clark and Chalmers, 1998; Risko and Gilbert, 2016). This paper studies the adoption of coding agents across roles and sectors, and estimates effects of these agents on output. Agents may help to offload the syntactic activity of writing code, and shift work patterns toward communicating logic in natural language and evaluating generated outputs.

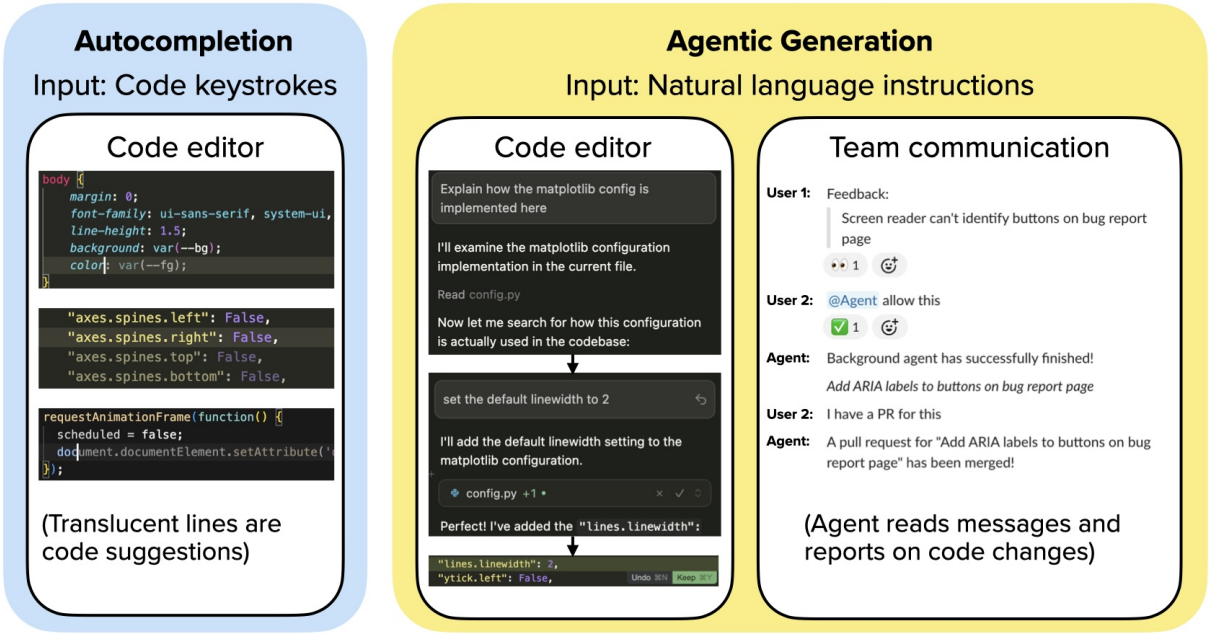
This paper contributes to the literature studying AI in software development. Field experiments (Cui et al., 2024) and task-specific experiments (Peng et al., 2023; Paradis et al., 2024) find shorter task completion times for people who use AI-assisted programming, particularly among less experienced developers. Becker et al. (2025) randomize AI availability across tasks among 16 developers and find longer completion times for AI-use tasks. Cross-sectional evidence finds positive correlations between AI use, software production speed, and bugfix rates among organizations (Arcolano, 2025), and variation in chatbot and agent use across message styles, use cases, and implied project types (Anthropic, 2025). Hoffmann et al. (2024) find that developers eligible for AI-assisted coding tools shifted their work toward coding outputs and away from project management outputs. Chen and Stratton (2025a,b) study productivity effects of AI-assisted programming tools across detailed categories of output tasks. This paper studies the impacts of agents on software development. By using a natural experiment induced by a coding agent's release timeline, this paper can identify effects of agents in a field setting. This field design observes developers' continual use of agents in their jobs, which allows for longer-term measurement than laboratory experiments on AI use. This paper introduces statistics on what kinds of people use AI tools, how people use agents, and how agent usage patterns relate to work experience. Finally, this paper uses

categorizations of natural language message data to study the experience of working with agents.

This paper also contributes to the literature on how people interact with AI outputs in the field. In prediction policy problems (Kleinberg et al., 2015), people respond to the outputs of algorithms that predict specific outcomes (e.g. Agrawal et al., 2018; Angelova et al., 2023; Stevenson and Doleac, 2024). Coding agents are not targeted to predict specific outcomes; people instead use these tools to generate new code sequences. Users can influence how these algorithms behave by providing context: People may change their interaction patterns as algorithms evolve (Jahani et al., 2025), and steerable algorithms may call for new design patterns (Vafa et al., 2025). The choice of which context to provide may influence alignment. A related literature studies usage and effects of voice agents (Choi et al., 2025; Jabarian and Henkel, 2025). This paper studies how people interact with agentic AI systems and apply generations from agents to output.

Finally, this paper contributes to the broader literature on AI's economic effects, which includes work studying AI and productivity (e.g. Noy and Zhang, 2023; Brynjolfsson et al., 2025b), the growing footprint of AI use (e.g. McElheran et al., 2024; Bick et al., 2024; Humlum and Vestergaard, 2025; Chatterji et al., 2025), interaction patterns with language models (e.g. Zhao et al., 2024; Handa et al., 2025; Tomlinson et al., 2025; Chatterji et al., 2025), AI as cognitive automation for research (Korinek, 2023), market effects of agents (Rothschild et al., 2025; Shahidi et al., 2025; Hadfield and Koh, 2025), worker preferences for agents (Shao et al., 2025), and agents as incomplete representations (Liang, 2025). Labor market effects of AI tools may depend on how technical capabilities and task-level applications relate to worker expertise (Garicano and Rossi-Hansberg, 2006; Autor and Thompson, 2025), changes in demand for worker skills (Katz and Murphy, 1992), and reallocation of tasks within roles (Hampole et al., 2025). Ide and Talamàs (2025) argue that non-autonomous AI systems may complement workers with less expertise and autonomous AI systems may complement workers with more expertise. Recent hiring for junior workers has differed across occupations and firms—some researchers argue these differences may stem from AI exposure (Brynjolfsson et al., 2025a; Lichtinger and Maasoum, 2025), and other researchers find smaller aggregate reallocation (Gimbel et al., 2025). The effectiveness of AI applications has varied across tasks (Dell'Acqua et al., 2023; Dillon et al., 2025), and people do not always effectively infer how model performance generalizes across domains (Vafa et al., 2024). This paper analyzes the adoption and economic impacts of agentic AI systems, which integrate language models with external tools. Workers can use these systems to perform new

Figure 1: Example applications of AI-assisted programming.



Notes – This figure shows examples of AI-assisted programming across modalities.

- Autocompletion (left panel). As a user types code, the autocompletion feature suggests new code that a user can choose to accept.
- Agent conversation and edits (middle panel). A user asks a question, the agent searches the codebase for an answer, and the user follows up with an instruction. The user can choose to accept the generated code edits.
- Team communication and agent instructions (right panel). One user shares product feedback, and a second user instructs the agent to address the feedback. The second user reviews the generated code change, which is eventually merged into the codebase.

process tasks related to instructing and evaluating agents. In contrast to other kinds of AI tools, coding agents have been applied to output more frequently by experienced workers.

1. Context and Data

I describe software development and AI-assisted programming, and document the data sources used in this paper.

1.1. Software Development and AI-Assisted Programming

Software development process Software development follows a lifecycle similar to product development in other contexts. An organization identifies problems, proposes solutions, and designs specifications for new software. Software developers write code to implement these designs, and test and verify the output of these code changes. Once software is finalized, it is deployed and maintained by each organization. Modern development often proceeds iteratively, with teams revisiting earlier stages of the lifecycle throughout.

To implement software, developers often use code editors. Code editors are programs that integrate text editing with other features. Feature-rich code editors can support debugging, error detection, and version control integration. In addition to code editors, developers also produce software using command-line tools, visual programming environments, and other code generators.

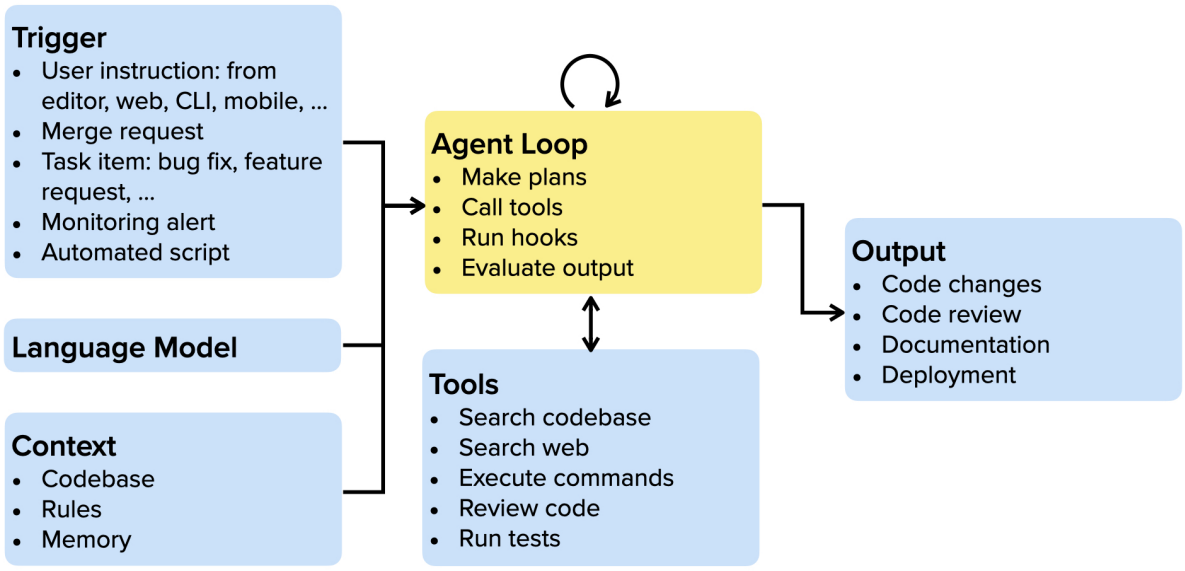
Organizations track code changes using version control, and apply both manual and programmatic review before deploying proposed changes. Version control systems record modifications to a codebase over time. These systems allow developers to work on parallel versions of a codebase for different work items, revert problematic changes, and collaborate without overwriting each other. Once a developer has finished a work item, they typically submit their code for review by other developers and automated verifiers. Reviewed code changes are then merged into the main codebase.

Once code changes are reviewed and merged, organizations deploy the overall codebase to production. With the growth of continuous integration and continuous deployment pipelines, code changes are being deployed to production at increasing frequency. If a deployment results in a failure—for example through degraded service or an outage, it is typically rolled back or hotfixed by the organization.

Software development productivity is typically measured using metrics related to speed, quality, system health, collaboration, and satisfaction. Forsgren et al. (2019) discuss metrics related to development speed (deployment frequency, lead time for changes), and development health (change failure rate, time to restore services). Forsgren et al. (2021) discuss holistic benchmarks that measure satisfaction, performance, activity, communication, and efficiency.

AI and software development Software development has been a common domain for AI applications. Code includes repetitive patterns (Hindle et al., 2016) that language models can recognize and apply. Many software outputs can be verified with tests,

Figure 2: Agents integrate language models with tools.



Notes – This figure describes how agents integrate language models with tools and context to produce output.

and such verification is useful for training language models for software development (Chen et al., 2021; Le et al., 2022; Chen et al., 2022). Modern software development also makes extensive use of code review and continuous integration (e.g. Martin, 2012; Sadowski et al., 2018; Fowler, 2024)—these validation practices may control the quality of AI-assisted code that ultimately gets deployed. In addition, people who produce AI-assisted programming platforms may regularly use such platforms as they work, which can speed up the production feedback loop.

Autocompletion and generation Two common modalities of AI-assisted programming are autocompletion and generation. Autocompletion is triggered by code editing keystrokes, while generation is triggered by natural-language messages.

Autocompletion suggests new code as users type into a code editor. Early autocompletion tools performed completion of matching code phrases—for example a user could press tab to finish typing a variable name previously defined in the codebase. Subsequent tools performed language model-based autocompletion, which can generate code sequences that may differ from existing patterns in the codebase.

Generation produces code from natural-language conversations with users. Early generation tools, like program synthesizers, transformed high-level specifications into simple programs. Subsequent tools performed richer generations that built on language model outputs. With a natural language interface, a user does not need to type in specific initial code sequences for a generation tool to produce code.

Agentic generation Recently, generation tools have implemented agentic capabilities. An agent takes a natural language instruction from a user and generates code by combining a language model with tools that can read files, run tests, access APIs, and conduct other tasks. Agents can break up user instructions into subtasks, and interact with external tools to generate code changes. Figure 2 describes components of coding agents.

In controlled environments, agents have increased performance across evaluation benchmarks, but have struggled with consistency and instruction-following (Liu et al., 2024; Jimenez et al., 2024; Yang et al., 2024). Quantifying the effects of agents in practice calls for empirical analysis of agent usage and output.

1.2. Data

To measure the adoption of AI-assisted programming and quantify agent usage patterns, I analyze aggregated usage metrics and professional characteristics from a random sample of 1,000 organizations. The sample is uniformly drawn from a subset of organizations with 50 or more users, with a recorded sector, and which had started using Cursor before July 15, 2025. To measure organization-level output, I analyze historical merge metadata from 32 organizations. For more detailed analyses, I analyze person-level activity, merge, and deployment data from one organization, which will be referred to as Company 1. For the analysis of message intent and work patterns, I analyze message intent classifications from a random sample of 1,000 individual users. All statistical analyses using these data report anonymized and aggregated estimates.

Adoption and usage styles To measure the adoption of AI-assisted programming, I analyze usage and professional characteristics from the 1,000-organization sample. The analysis focuses on a subset of 86,665 users matched to role, seniority, and years of work experience. The procedure for labeling professional characteristics from de-identified work history summaries is described in Appendix A.1. Statistics reported using professional characteristics have a minimum threshold of 100 users. To measure

adoption of the agent feature, I analyze weekly usage of the platform. Weeks are indexed on Mondays. To measure usage styles, I use a cross-section of this sample filtered to users with agent edit outcomes recorded during the final full week of the sample, which started on July 7, 2025. The analysis focuses on users that sent at least one agent message from the code editor with edit outcomes observed in that week. This results in a match to 44,160 users for this analysis.

Output, quality, and scope proxies To measure aggregate effects on output, I use a panel of organization-level merge metadata from October 2024 to May 2025. The metadata contains merge week, number of lines edited, number of files touched, and categorizations into reverts and bugfixes. I compare outcomes between an "eligible" group of organizations that started using Cursor before the analysis period began, and a "baseline" group of organizations that did not start using Cursor until after the analysis period ended. Since the eligible group was already using the platform when the analysis period began, its production technology would have been affected by the agent release and agent default updates. Meanwhile, the baseline group's production technology would not have been affected by these updates during the analysis period. There are 24 eligible organizations and 8 baseline organizations, with 323,589 merges covered over the 34-week sample period. To measure developer-level associations between agent usage and output, I analyze merge and deployment metadata from Company 1. This metadata is used for cross-sectional analyses that relate agent usage styles to outcome metrics across developers.

Semantic shift in work patterns To analyze the semantic content of agent messages, I analyze intent labels of initial messages in an agent conversation. The algorithmically-generated intent labels come from users on individual subscriptions who opted into sharing messages with the platform. The empirical analysis relates shares of these labels to work experience and accept rates. I analyze a random sample of labels from May and June 2025 across 1,000 users on individual subscriptions, uniformly drawn from a subset of individual users matched to years of work experience with at least 20 initial messages over the period. This corresponds to a sample of 77,024 message labels.

2. Adoption and Usage Patterns

AI-assisted programming has been adopted across sectors and roles. The sample consists of organizations that produce output including developer tools, financial tools, and media. The majority of sampled users are software developers, with a tail of usage across workers in roles including data, product management, and design. The platform is used across levels of seniority and experience.

Agent use has become a common modality of AI-assisted programming—a majority of sampled platform users adopted the agent feature in the six months following its release. In addition to software engineers, people working in lower-code roles like designers and product managers also use agents. Agent use is also frequent in lower-code sectors, including in consulting and professional services.

Developers with more years of work experience accept generated outputs more often. The least experienced decile of workers uses agents frequently, but is less likely to accept generated outputs.

2.1. User Characteristics

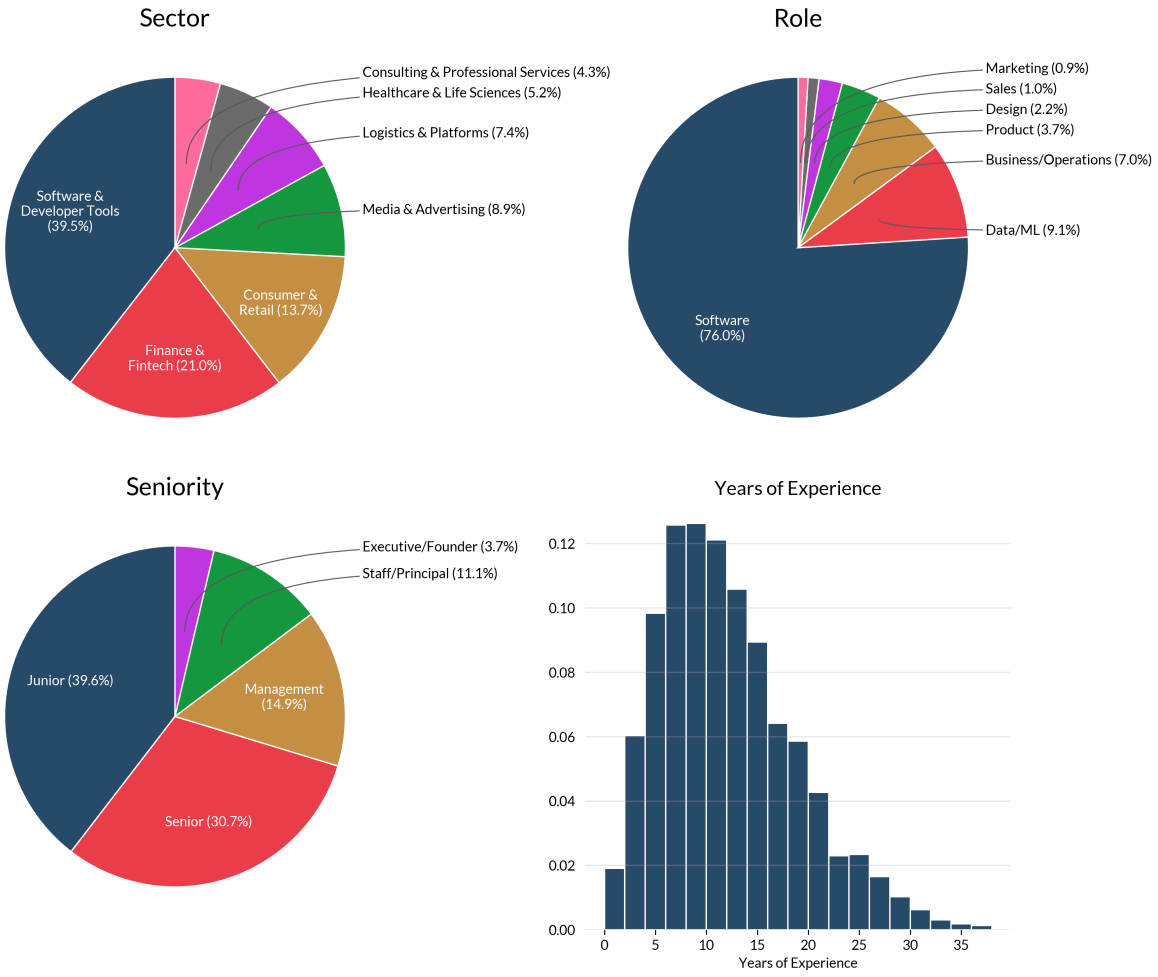
Figure 3 plots statistics on platform users across sectors, roles, seniorities, and years of experience. The top panels and bottom left panel plot the share of users in the 1,000-organization sample across each category. The bottom right panel plots the distribution of users across years of work experience.

AI-assisted programming is used across sectors. 39% of users in the sample work in organizations that produce software and developer tools. A further 44% work in finance/fintech, consumer/retail, and media/advertising (which includes gaming).

While the majority of users hold software roles, the platform is also used in roles with traditionally lower programming intensity. Software roles account for 76% of users in the sample, with data and machine learning roles accounting for a further 9%. Additional roles include product management, design, and sales.

40% of users hold Junior roles, 31% hold Senior roles, and 11% hold Staff/Principal roles (“Staff” and “Principal” are more senior roles in software development). A further 15% hold management roles, while 4% hold founder and executive roles. Adoption is highest across people with 5–15 years of work experience.

Figure 3: User characteristics.



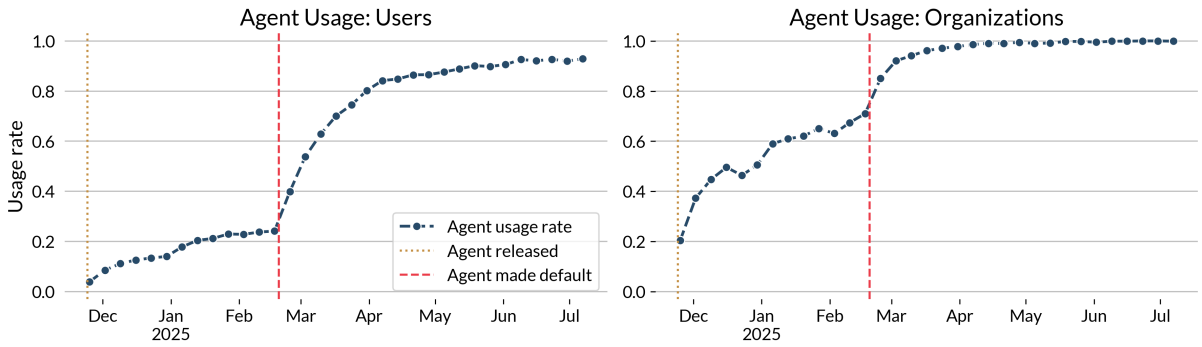
Notes – This figure plots the distribution of users in the sample of organization users across sectors, roles, seniorities, and years of work experience.

2.2. Diffusion of Agent Use

The platform's agent feature was released on November 24, 2024. Prior to the release of the agent feature, the platform included a tab autocompletion feature and a chat interface with standalone language models for generation. When the agent feature was initially released, users would have to manually select it to use agentic capabilities for generation. On February 19, 2025, the agent feature was made the default generation mode in the updated Cursor client.

Figure 4 plots the shares of active organizations and users in the sample who used the feature in a given week. In the week before the feature was made default, 24% of

Figure 4: Diffusion of active agent use in the sample.



Notes – This figure plots the share of organizations and users who used the agent feature, relative to all active platform-using organizations or users in the sample in a given week. The feature was released in the November 24, 2024 update, and was made the default generation mode in the February 19, 2025 update. The agent would become the default setting when users updated their software clients.

active users and 67% of active organizations in the sample used the agent feature. At the end of the sample period, 92% of active users and all active organizations in the sample used the agent feature.

2.3. Usage Styles

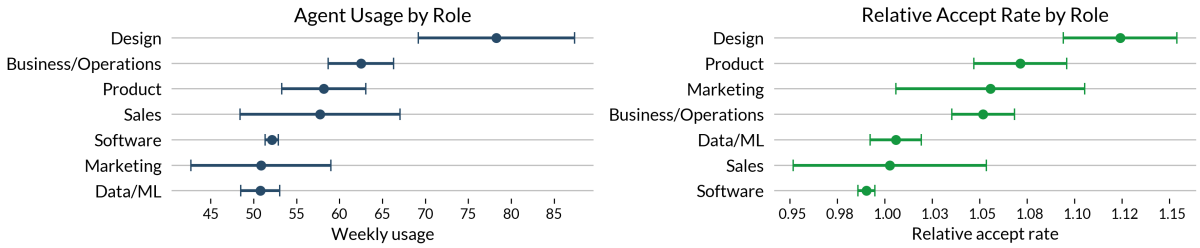
People across roles and sectors use agents. Experienced users are more likely to accept generations from agents. The least experienced decile of users uses agents frequently, but is less likely to accept generated output.

Figure 5 plots agent usage styles across roles, sectors, seniorities, and years of experience. It includes statistics on agent message frequency and agent accept rates. Agent message frequency is the number of agent messages submitted from the code editor. Relative accept rate is the share of agent messages with recorded edit outcomes for which an edit was accepted, scaled by the mean accept rate.

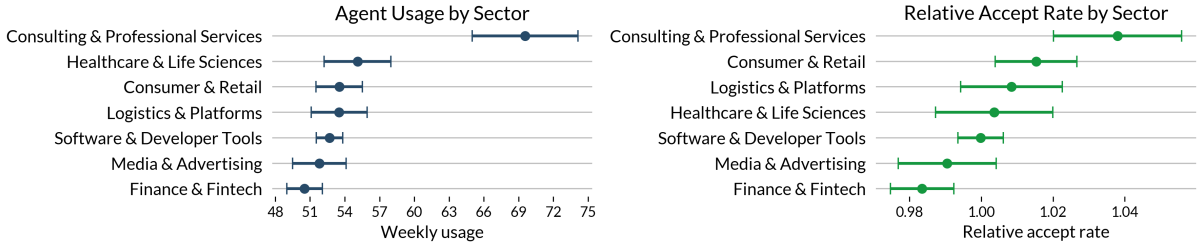
Usage frequency, accept rates, and task types Figure 5a plots usage styles across user roles. Software engineers submit 52.1 agent messages per week. Agents are also used by workers in non-engineering roles. Designers submit 74.7 agent messages per week, and accept 1.14 times as often as software engineers do. Product managers submit 56.3 agent messages per week, and accept 1.08 times as often as software engineers do. For non-engineering roles, agents may assist with prototyping designs and features.

Figure 5: Agent usage styles by user characteristics.

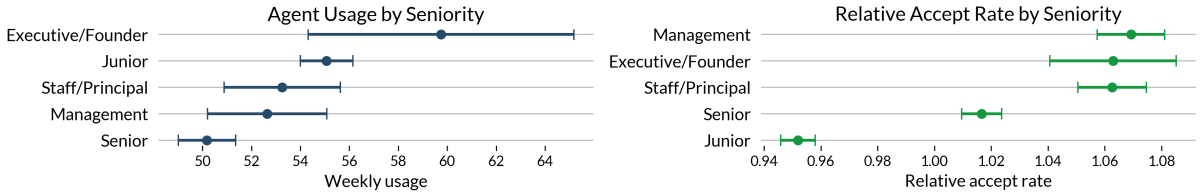
(a) Role



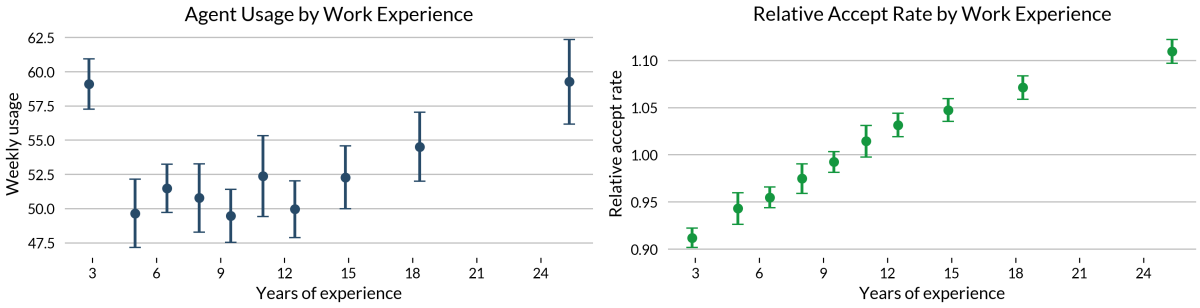
(b) Sector



(c) Seniority



(d) Years of Work Experience



Notes – This figure plots agent usage patterns across role, sector, seniority, and decile of work experience from the agent metrics sample. Weekly usage corresponds to total agent messages. Relative accept rate is the share of messages with recorded edit actions for which a code edit was accepted, scaled by the mean rate. Error bars report 95% confidence intervals for group means.

People may be more likely to delegate these tasks, which are verifiable at short horizons, to agents. If agent use increases among non-engineering roles, some of these tasks may be reallocated from engineering to non-engineering roles.

Figure 5b plots usage styles across sectors. Workers at consulting and professional services firms have the highest agent usage frequency (68.0 messages per week) and accept rate. Software outputs at these firms could include prototypes and dashboards, which may be verifiable at short horizons. Workers at consulting firms may also have more liberal accept criteria, given their output may be less likely to integrate into complex systems. Workers at finance and fintech firms have lower agent usage frequency (49.1 messages per week) and accept rates. Codebases at financial firms may be more bespoke and code changes may require more precise verification.

Figure 5c plots usage styles across seniorities. Users with management titles submit 50.6 messages per week and have the highest accept rates. Users with junior titles submit 53.8 messages per week and have the lowest accept rates.

Experience gap in agent accept rates Figure 5d plots usage styles across years of experience. Experienced users are more likely to accept agent-generated code. The least experienced decile of users is more likely to submit agent messages, but is less likely to accept generated code edits.

The experience gap in agent accept rates persists after adjusting for role, sector, and seniority. Table A4 reports the results of regressions of accept rate on work experience. After adjusting for role and sector, one standard deviation (6.6 years) higher experience corresponds to 5.9% higher accept rates relative to the mean. After further adjusting for job title seniority, one standard deviation higher experience corresponds to 4.9% higher accept rates relative to the mean. Table A5 further adjusts for total usage. While higher usage is associated with higher accept rates, higher experience continues to be associated with higher accept rates. The positive experience gap for agent accepts contrasts with a negative experience gap for autocompletion accepts. Figure A6 shows that less experienced workers are more likely to accept suggestions from the autocompletion feature.

Interpreting accept rates and sample composition The decision to accept an agent generation depends on the alignment between agent output and user intent, the types of tasks a user conducts with agents, and a user's threshold for accepting generated code. Alignment may vary with a user's skill providing instructions to agents, including through

clarity of description and effective mental models of agent capabilities. Task distributions may also vary—an alternative explanation for these results is that experienced workers perform tasks that agents are more suited for. If this task distribution shift is with respect to process tasks (i.e. senior workers break up equivalent business features into chunks better suited for agents) instead of output tasks (i.e. senior workers work on business features that are better suited for agents), higher accept rates may still be consistent with higher skill using agents. Unconditional accept thresholds may vary with the value of aligned output to each individual. For example, less experienced workers may be newer team members who are more conservative acceptors of agent-generated output.

Further evidence is consistent with a skill-based explanation for the gap. A task distribution-based explanation would imply that experienced workers perform tasks better suited for agents. However, Table A4 finds that the experience gap in accept rates persists after adjusting for role, sector, and seniority—the task distribution-based explanation would require selection on unobservables not explained by these characteristics. A threshold-based explanation would imply that experienced workers have incentives or usage styles that drive up accept rates. If experienced workers conduct higher-value work with higher costs of failure, it would be unlikely for such a relationship to generate higher agent accept rates. If experienced workers were less vigilant agent users, they might be less likely to make plans in their initial messages, which would run counter to the results in Section 4. If experienced workers are less risk-averse, they might have higher agent accept rates than less experienced workers, but would be unlikely to have lower autocompletion accept rates as reported in Figure A6 unless risk perceptions differ between tools. If autocompletion helps workers complete tasks, while agents help workers delegate tasks, it is possible that experienced workers currently have more skill specifying delegated tasks and evaluating generated outputs.

These agent usage and accept statistics are conditional on a worker's choice to adopt the platform and submit messages to agents. Usage patterns may change as adoption patterns change. For example, while designers have the most frequent agent usage, designers who currently use AI-assisted programming tools may be more active in software development than most designers at software organizations. Nonetheless, these statistics correspond to usage patterns within the sample of adopting users. To measure organization-level effects, I turn to an analysis of organization-level outcomes.

3. Output

The release of the agent feature was associated with increases in software output, and no significant changes in a short-run software quality proxy. Section 3.1 reports organization-level difference-in-differences estimates across historical merge metadata from 32 organizations. Using variation from a natural experiment induced by the agent feature's release timeline, I estimate effects of the agent release on merges, lines edited, revert rates, and bugfix rates.

Frequent agent users produce more output—they merge more code, edit more lines, and edit more files per merge. Section 3.2 reports person-level correlational evidence on agent usage and software development outcomes from one organization. As this second set of results is not the product of a natural experiment, the evidence could reflect a combination of effects of agent use and characteristics of workers who are more productive or more open to new technologies.

3.1. Output: Organization-Level Evidence

Measurement Using the weekly panel of merge metadata described in Section 1.2, I analyze output and quality metrics in the weeks surrounding the release of the agent feature. These are historical outcomes observed at the organization level.

As the main measure of output, I use the rate of weekly merges to the main codebase. Merges are semantic units of work—a developer submits a merge request to address a specific goal. While merges can be interpreted as economic units of work, their impact is not uniform—some code merges may address more valuable business goals than others. However, since merges are units of code output used by all organizations, they allow for aggregate analysis of code output.

As an additional measure of output, I include weekly lines edited. Lines edited per merge are winsorized at the 1% level at the right tail, and then summed across merges each week. Edited line count is a syntactic metric, and varies with programming language verbosity. It may also increase if agents produce more code comments or more verbose code for similar logic. Like merges, lines edited are units common to all organizations.

As the main proxy for quality, I use the weekly share of merges that are labeled as reverts. The revert rate is a proxy for the frequency of breaking code changes across a codebase. An advantage of this metric is that it indicates a likely severe issue—for example, an organization may revert a merge if it creates degraded service in deployment. However, the revert rate may oversample high-visibility, short-horizon issues.

As an additional proxy for quality, I use the weekly share of merges that are labeled as bugfixes. The bugfix rate is a proxy for the share of work spent fixing software issues across a codebase. An advantage of this metric is that it corresponds to issues in a codebase that might not be caught immediately after an initial merge. Agent use may affect the bug fixing behavior of organizations in other ways—for example by shifting the focus of engineers from low-priority bugfix work to new feature work.

Both these measures are short-term proxies for quality. If agent-produced code creates technical debt, it may take longer to measure negative effects on quality. If agent-produced code is more standardized and easier to maintain, it may take longer to measure positive effects on quality. Furthermore, these proxies do not measure other quality-relevant patterns, including duplicated logic across the codebase and code complexity.

Estimated effects of the agent feature To quantify effects of the agent feature release, I estimate dynamic difference-in-differences regressions following the specification

$$Y_{i,t} = \alpha_i + \alpha_t + \sum_{k \neq -1} \beta_k \times D_{i,t,k} + \varepsilon_{i,t} \tag{1}$$

across organizations i and calendar weeks t. $D_{i,t,k} = 1$ if organization i is in the eligible group and week t is k weeks relative to the release week. α_i and α_t are organization and week fixed effects. This difference-in-differences design can adjust for time trends that unconditionally affect software-producing organizations—for example, lower output on holidays.

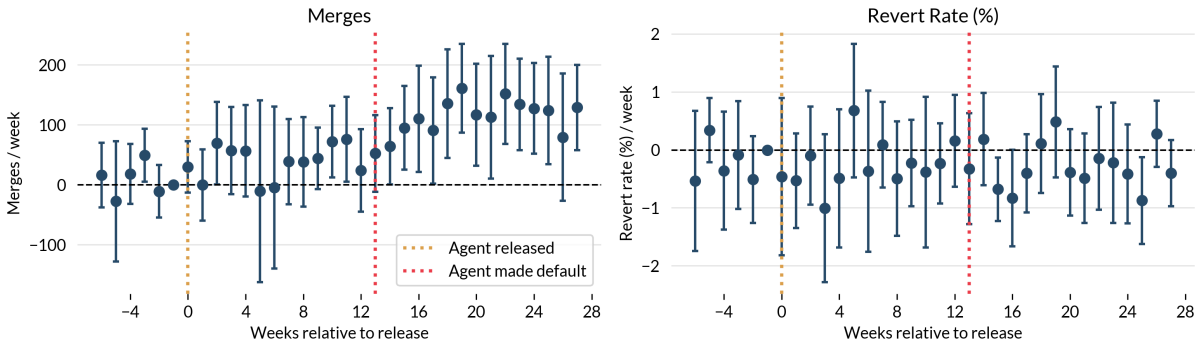
Figure 6 reports results from these regressions across the two main outcomes for output and quality. In the Appendix, Figure A7 reports results across additional outcome variables, Table A8 reports pre-period balance statistics, and Figure A9 plots weekly mean values across periods and groups.

To quantify average effects, I compare changes in outcomes between the eligible and baseline organizations. I use the difference-in-differences specification

$$Y_{i,t} = \alpha_i + \alpha_t + \beta \times \text{Eligible}_i \times \text{Post}_t + \varepsilon_{i,t} \tag{2}$$

In one set of specifications, I compare outcomes on and after the release week to outcomes before the release week. In another set of specifications, I compare outcomes on and after the default week to outcomes before the release week, filtering out the

Figure 6: Estimated effects of the agent feature.



Notes – This figure plots dynamic difference-in-differences estimates of merges and revert rates among organizations eligible for the agent feature. The estimates follow the specification in Equation (1). Vertical lines denote the weeks of the release date and default date of the agent feature. Standard errors are clustered by organization.

intermediate period.

Table 7 reports the results of regressions following Equation (2) across the four outcomes. Over the period after the agent release, Table 7a indicates that merges in the eligible group were 26% higher and lines edited were 29% higher relative to the pre-release period.[1] Over the default period, Table 7b indicates that merges in the eligible group were 39% higher and lines edited were 49% higher relative to the pre-release period. Table 7 also shows no significant changes in revert rates and decreases in bugfix rates. Table A11 reports the results of additional inference using the wild cluster bootstrap-t.

Merges and productivity: Scope The results from Figure 6 and Table 7 indicate that merge output increased after the agent was released and made the default generation mode. Can these changes in merge output be interpreted as increases in productivity?

An alternative explanation of these results would be that agents cause developers to produce smaller-scoped changes in each merge. Such an explanation would also be consistent with a higher merge rate. However, if this alternative explanation caused an increase in merges, there would be additional implications on merge characteristics that do not appear consistent with the data.

Table A12a reports estimates of the effects of the agent release on proxies for merge scope: Lines edited per merge and files touched per merge. The analysis finds no

[1] In the eligible group, the pre-release sample mean for merges is 271.7 and for thousands of lines edited is 50.5.

Table 7: Agent availability is associated with higher output.

(a) Post-release (full) versus pre-release.

	Merges (1)	Lines Edited (K) (2)	Revert Rate (%) (3)	Bugfix Rate (%) (4)
Eligible × Post Release	70.09	14.80	-0.07	-1.41
	(18.60)	(4.91)	(0.14)	(0.58)
R^2	0.87	0.83	0.27	0.53
Within R^2	0.02	0.01	0.00	0.00
Observations	1,088	1,088	1,088	1,088
Organization FE	✓	✓	✓	✓
Time FE	✓	✓	✓	✓

(b) Post-default versus pre-release.

	Merges (1)	Lines Edited (K) (2)	Revert Rate (%) (3)	Bugfix Rate (%) (4)
Eligible × Post Default	104.69	24.61	-0.08	-2.40
	(25.54)	(7.48)	(0.17)	(0.70)
R^2	0.90	0.87	0.37	0.66
Within R^2	0.05	0.05	0.00	0.02
Observations	672	672	672	672
Organization FE	✓	✓	✓	✓
Time FE	✓	✓	✓	✓

Notes – These tables report difference-in-differences estimates of outcomes related to output (merges and thousands of lines edited) and quality (revert rate and bugfix rate) among organizations eligible for the agent feature. The estimates follow the specification in Equation (2). The top panel compares outcomes from the post-release and pre-release periods. The bottom panel compares outcomes from the post-default and pre-release periods. Standard errors clustered by organization are reported in parentheses.

significant changes in these scope proxies over the sample period. Confidence intervals rule out changes large enough to offset the increase in merges. Given these results, the alternate explanation would require further assumptions on the effects of agent use: Agent use would need to have produced verbose code edits that increased lines edited without increasing scope—such edits could keep lines edited per merge consistent while decreasing the scope of each edited line. Even if this further assumption holds, the

alternative story would also have to account for no significant decreases in files touched from agent use. Such an assumption would require not only higher agent verbosity over edited lines, but also more frequent agent edits over extraneous files. Developers would have then chosen to merge these extraneous files. According to the evidence on these proxies, merge scope does not appear to have decreased with agent use.

Merges and productivity: Value What kinds of work does agent use affect? Table A12b considers changes in merge output across small and large merges, defined as those below and above the median of lines edited or files touched, with respect to eligible organizations during the pre-release period. I estimate effects on these merge types both in the period between release and default and in the period after default. I find similar effects across merge sizes, which suggests that agent use increases output across merge scopes. If merge scope relates to merge value, these results suggest that the marginal value of merges from agent use is no lower than the value of merges in the pre-release period. However, more comprehensive estimates of value may require longer horizons and organization-specific outcome measures.

Internal and external validity These estimates reflect changes in outcomes for organizations in the sample. How generalizable are these results?

One set of questions relates to internal validity. A challenge to identification would be that a contemporaneous change in some other driver of code merges—which differentially affected the eligible group relative to the baseline group—coincided with the agent's release and default dates. There were no other Cursor client updates in the months of the agent release and default dates, which makes it unlikely that another platform event would affect the eligible group around those dates. The platform also did not announce these dates in advance, which makes it unlikely that organizations adjusted merge behavior in anticipation of agent releases. Each organization in the eligible group had already been using the platform for at least six weeks before the agent was announced and released, and the platform did not differentially target feature rollouts according to organization characteristics. Given these institutional details, it is reasonable that the agent's release and default generated exogenous variation in the eligible group's production technology.

Another set of questions relates to external validity. If the early adopters of Cursor who make up the eligible group are experimental organizations that are receptive to new technologies, the estimated effects on this group may be higher than analogous

effects of agents on other groups. Such an "experimental organization" characterization is consistent with the eligible group's higher pre-period revert rates according to the balance statistics in Table A8. However a heterogeneity analysis in Table A13 indicates that eligible organizations with higher pre-period revert rates had no larger increases in merges after the agent was released and made default. Moreover, Table A8 indicates that the organizations in the eligible and baseline groups are similar on observables other than the revert rate, including merge frequencies and bugfix rates. The similarity along these balance statistics suggests that other potential confounders, like codebase maintainability, would not be key drivers of cross-group effect heterogeneity in this sample. The effects of agents on output may also increase with time as the technology matures and as organizations invest in complements (e.g. Brynjolfsson et al., 2021).

3.2. Output: Person-Level Evidence

To measure person-level correlates of agent use, I analyze data from Company 1 on merges and deployment outcomes from May and June 2025. Merges and lines edited have the same definitions as in Section 3.1. I also use review time, defined as the hours between a merge request's creation and the final codebase merge. Rollback rate corresponds to the share of merges by a user that were attributed to any rollback in deployment. I also consider unique files, unique filetypes, and unique codebase areas edited per merge as proxies for scope.

To estimate cross-sectional relationships between agent usage frequency and outcomes, I use the specification

$$Y_i = \alpha + \beta \text{Usage}_i + \varepsilon_i \tag{3}$$

Table 8 reports the results of these regressions.

Higher agent use is associated with higher output. One standard deviation higher agent use is associated with 5.13 more merges and 2,940 more lines edited over the two-month period. Moreover, one standard deviation higher agent use is associated with 0.61 more files touched per merge and 0.08 more filetypes touched per merge, which could indicate larger-scoped code changes.

These results reflect cross-sectional evidence and, unlike the results from Section 3.1, are not a product of a natural experiment. Therefore, while the cross-sectional results could reflect consequences of agent use, they could also reflect characteristics of output from workers who are more disposed to use agents. For example, more productive

Table 8: More frequent agent usage is associated with higher output.

(a) Output and review velocity.

	Total Merges (1)	Total Lines (K) (2)	Lines (K) / Merge (3)	Review Time (h) (4)
Agent Usage (σ)	5.13	2.94	0.04	1.75
	(0.96)	(0.44)	(0.02)	(1.52)
Constant	30.87	9.79	0.39	43.96
	(0.90)	(0.44)	(0.02)	(1.66)
R^2	0.04	0.05	0.01	0.00
Observations	783	783	783	783

(b) Quality and scope.

	Rollback Rate (%) (5)	Files / Merge (6)	Filetypes / Merge (7)	Areas / Merge (8)
Agent Usage (σ)	-0.06	0.61	0.08	0.01
	(0.21)	(0.20)	(0.02)	(0.02)
Constant	3.61	6.26	1.68	1.19
	(0.26)	(0.19)	(0.02)	(0.02)
R^2	0.00	0.01	0.02	0.00
Observations	783	783	783	783

Notes – These tables report the results of cross-sectional regressions of outcomes on agent usage frequency. The estimates follow the specification in Equation (3). One standard deviation in usage corresponds to 459 messages over the two-month period. The top panel reports results on outcomes corresponding to throughput and velocity. Columns (1) and (2) correspond to total outcomes per person, and Columns (3) and (4) correspond to merge-level means per person. The bottom panel reports results on outcomes corresponding to quality and scope. Column (5) corresponds to a rate per person, and columns (6) to (8) correspond to merge-level means per person. Robust standard errors are reported in parentheses.

workers may be more willing to use new technologies like agents, and may work on larger-scoped code changes.

4. Semantic Shift in Work Patterns

People use agents by communicating in natural language—this usage style may influence the experience of programming. An intent classification of messages to agents indicates

Table 9: Example messages and labels

Message	Label
Can you do a codebase search for `Function` and restrict to `Location` and `Location`.	Explain Code/Codebase
Error: `Message` why this happens in `Location` ?	Explain Error
Implement this `Class`, using the `Class` in @`Location` as a reference	Implement
Do we need the `Service` in here? Is there a better pattern for this we can follow? Can we abstract anything here into helpers?	Plan
Where is `Object` constructed for the backend? What address does it hit?	Explain Code/Codebase
`Service` is already faster at around 3/4 the time of the original implementation. List some ideas we can make this even faster.	Plan
generate new @`Location` for all pages changed in @Branch	Implement
can you refactor this and get rid of `Config` and instead use individual arguments for each function	Implement
why is the `Object` closing when one is toggled? don't want that to happen	Explain Error
What does this file do in the context of the project	Explain Code/Codebase

Notes – This table presents examples of initial messages to agents along with intent labels. The messages were sent by developers at Anysphere, the company that releases Cursor, from their internal accounts. Specific codebase information is replaced with generic labels.

that people use agents to implement code, develop plans, explain errors, and explain code. These behaviors offload different parts of the software development process. More experienced workers are more likely to send planning instructions in their first messages, which may improve alignment. People with plan-first usage styles are more likely to accept agent edits.

4.1. Message Intent

To analyze message styles, I use a labeling of messages into user intent. This labeling consists of categories for "Implement," "Plan," "Explain Code/Codebase," "Explain Error," and "Other." More details on the labeling procedure are presented in Appendix A.1. For illustration, Table 9 includes example messages from a separate sample of developers at Anysphere using their internal accounts, along with model-implied labels.

For the empirical analysis, algorithmically-generated labels are derived from the sample of opt-in individual users described in Section 1.2. The empirical analysis studies shares of these labels.

Intent labeling "Implement" messages are instructions to change code and configurations. They can include feature work, bugfixes, test setup, and content generation. "Plan" messages are instructions to specify code changes, and requests for guidance on implementing software. They can include requests for pros and cons, recommended options, and step-by-step plans. "Explain Code/Codebase" messages are requests to interpret existing code, system logic, and library functionality. They can include build and log outputs to analyze behavior. "Explain Error" messages are requests to explain errors. They can include logs and stack traces, and requests about likely causes of errors. "Other" messages do not fit into this taxonomy, and include greetings and non-code-related queries.

While non-implementation messages do not clearly request implementation, they may still trigger code generation by an agent. For example, an "Explain Error" message may generate a suggested code change from an agent to fix a bug. Message content also does not include editor context, which may make an implementation intent clearer to an agent. For example, the "Explain Code/Codebase" message "How does function A handle negative input values?" could be interpreted by the agent as an implementation request if a user is working on the function and did not add logic for handling negative input values.

These results reflect behaviors of platform users with individual subscriptions, which may differ from behaviors of users with organization subscriptions. For example, development in this sample may be skewed toward personal projects, which may involve more exploration and greenfield work. The distribution of work experience also has a higher mean and is more dispersed in the individual sample than in the organization sample.[2] However, if the behavioral patterns correlated with work experience are similar between the organization and individual samples, the relationship between work experience and intent may be similar in both samples.

Agents, planning, and offloading Figure 10 reports the distribution of initial messages by intent label. 60.9% of messages have an "Implement" intent, 13.6% have a "Plan" intent, 12.1% have an "Explain Code/Codebase" intent, 11.9% have an "Explain Error" intent, and 1.5% have an "Other" intent. In addition to using agents to translate specifications into code, people also use agents to make plans, explain errors, and explain code.

[2]The mean years of work experience in the individual user sample is 14.2, and the standard deviation is 8.4. The 25th percentile is 7, the median is 13, and the 75th percentile is 20.

Figure 10: Distribution of message intent



Notes – This figure plots the distribution of intent in conversation-starting messages to agents from the individual user sample. 60.9% of messages have an "Implement" intent, 13.6% have a "Plan" intent, 12.1% have an "Explain Code/Codebase" intent, 11.9% have an "Explain Error" intent, and 1.5% have an "Other" intent.

Agents may alter the behaviors that developers have traditionally used to understand code. For example, a developer might substitute time spent searching through a code library's documentation toward time spent asking agents how the library works. Developers might also substitute some of their codebase explanation questions away from coworkers and toward agents.

The use of agents to explain implies that people may be offloading some parts of the production process to external representations (e.g. Zhang and Norman, 1994). Such offloading may aid production by outsourcing knowledge related to one-off tasks. It could also hinder production if, for example, a developer uses agents to address similar types of errors but does not internalize the source of the errors for future work.

The use of agents to plan implies that agents may influence how workers choose to allocate effort. People who use agents to generate plans may review and modify proposed steps before the agent acts. Such users may be more inclined to seek alignment with agents. Consistent with this view, Table A15a finds that users with higher rates of plan-first intents also have higher agent accept rates. Given planning requests can include instructions to weigh pros and cons, people may also delegate some judgment tasks to agents.

4.2. Experience, Context, and Alignment

As people gain work experience, their familiarity with codebase structure and interaction patterns with colleagues may change. These changes may influence how experienced workers interact with agents. To analyze the relationship between message intent and work experience, I estimate regressions following

$$\text{Intent Frequency}_i = \alpha + \beta \text{Work Experience}_i + \varepsilon_i \tag{4}$$

Table 11: Experienced workers are more likely to make plans in their initial messages.

	Implement (1)	Plan (2)	Explain Code (3)	Explain Error (4)	Other (5)
Work Experience (σ)	-0.68	1.63	-0.83	-0.24	0.12
	(0.61)	(0.40)	(0.35)	(0.24)	(0.09)
Constant	58.91	15.02	13.20	11.23	1.64
	(0.58)	(0.37)	(0.35)	(0.27)	(0.10)
R^2	0.00	0.02	0.01	0.00	0.00
Observations	1,000	1,000	1,000	1,000	1,000

Notes – This table reports the results of regressions of message intent frequency on work experience, following Equation (4). One standard deviation in work experience corresponds to 8.4 years of experience. Robust standard errors are reported in parentheses.

across users i. Table 11 reports the results of these regressions.

Experienced workers send more planning messages For every standard deviation increase in work experience, people are 1.6 percentage points more likely to send "Plan" messages. Experienced workers may be more likely to send plan instructions because they have had more experience communicating goals with colleagues. To the extent that experienced workers review and modify agent-produced plans, this behavior may improve alignment with user intent. Table A15b shows that both work experience and planning intent frequency are associated with higher accept rates.

Experienced workers send fewer code explanation messages For every standard deviation increase in work experience, people are 0.8 percentage points less likely to send "Explain Code/Codebase" messages. This evidence suggests that experienced workers may have more code context and codebase understanding.

Adjusting for total usage Table A14 reports regression estimates of the relationship between message intent and work experience that further control for total usage. Similar to the results from Section 3.2, frequent users' behavior appears skewed toward producing output. There are more messages related to implementation and explaining errors, and fewer related to explaining code and planning. After adjusting for total usage, experienced workers continue to send more "Plan" messages and fewer "Explain Code/Codebase" messages.

4.3. Higher-Order Thinking

Agents add a layer of abstraction to production. This move toward abstract production is not a new phenomenon. In software development, production has become more abstract as the programming environment has evolved from bit manipulation to assembly code to high-level programming languages. Many programmers have adopted additional layers of abstraction, including objects, garbage collection, and libraries.

Across other sectors, production has become more abstract as workers use video appointments to diagnose patients, digital workstations to produce music, and algorithms to trade.

Abstractions may increase scale and simplify cognition, but may also introduce overhead and reduce precision. To use abstractions effectively, people may benefit from understanding what the abstractions represent and when the abstractions have limits. Coding agents are abstractions that transform expressed user intent into code—this new abstraction may increase the value of specific skills.

Abstract thinking Effective agent use may depend on clear mental models of agent capabilities and task composition. A developer might improve alignment by using past experiences to forecast an agent's performance across new tasks. A developer who understands how to combine tasks to achieve a high-level goal might delegate to agents more effectively.

Clarity Using agents to implement output involves offloading part of the production process. Ambiguity in this offloading step could lead agents to produce output a user did not anticipate. Precise instructions may reduce such misalignment between intent and output.

Evaluation An agent may not translate goals into output in the same way a person would. A developer who manually types code may run into design considerations and implementation issues that influence the final output—the same developer may not engage in the same cognitive processes when an agent produces code. Evaluation that accounts for likely failure modes may reduce this alignment problem.

Discussion

Agent use is associated with higher output and natural language-driven work patterns. If agents help workers delegate tasks, non-specialists may perform some specialist tasks, workers with more organizational context may delegate tasks more successfully, output may increase across organizations, and work patterns may shift from executing tasks to specifying tasks.

The results in this paper are consistent with these four implications. While coding agents are primarily used by software developers, they are also used frequently by people working in non-engineering roles. Experienced workers are more likely to accept generated code, which is consistent with higher skill using agents. The release of a coding agent was associated with increases in software merge output. After the release, there were no significant changes in revert rates and decreases in bugfix rates, though effects on codebase structure and maintainability may require longer horizons to measure. People communicate with agents using natural language: They use agents not only to implement output, but also to explain code patterns and to make plans. Experienced workers are more likely to develop plans with agents before acting, which may improve alignment.

These results correspond to early evidence centered around software production. While software production generates quantifiable outputs that can serve as economic proxies, longer-run effects in this domain will require additional data to measure. In addition, software production contains syntactic regularities that AI agents can match, short-run verifiable outputs that can be useful for post-training, and organization-level integration practices that may build some safeguards for agent-generated output. While it is possible that agents may have similar effects in domains with similar properties—for example in mathematics, accounting, or law—evaluating such impacts will require additional empirical evidence.

Effective agent use may call for fluency with abstraction, clarity, and evaluation. People who use coding agents partially abstract from the process of typing code sequences. Applying these systems to output involves evaluating generations from agents. At the moment, experienced workers may have more fluency with these skills, given more years spent developing codebase context, communicating intentions to other workers, and evaluating code. This skill distribution may change as people learn to use agents and the underlying technologies evolve. Future research could evaluate the returns to investing in these skills as knowledge work becomes more abstract.

References

Acemoglu, Daron and David Autor, "Skills, Tasks and Technologies: Implications for Employment and Earnings," in "Handbook of Labor Economics," Vol. 4, Elsevier, 2011, pp. 1043–1171. 5

_ , Fredric Kong, and Pascual Restrepo, "Tasks At Work: Comparative Advantage, Technology and Labor Demand," Technical Report w32872, National Bureau of Economic Research, Cambridge, MA August 2024. 5

Agrawal, Ajay, Joshua Gans, and Avi Goldfarb, "Prediction, Judgment, and Complexity: A Theory of Decision-Making and Artificial Intelligence," in "The Economics of Artificial Intelligence: An Agenda," University of Chicago Press, 2018, pp. 89–110. 6

Angelova, Victoria, Will Dobbie, and Crystal Yang, "Algorithmic Recommendations and Human Discretion," Technical Report w31747, National Bureau of Economic Research, Cambridge, MA September 2023. 1, 6

Anthropic, "Anthropic Economic Index: AI's Impact on Software Development," April 2025. 5

Arcolano, Nicholas, "Jellyfish and OpenAI Team up to Measure the Impact of AI Coding Tools," August 2025. 5

Athey, Susan and Scott Stern, "The Impact of Information Technology on Emergency Health Care Outcomes," *The Rand Journal of Economics*, 2002, *33* (3), 399–432. 5

Autor, David and Neil Thompson, "Expertise," Technical Report w33941, National Bureau of Economic Research, Cambridge, MA June 2025. 6

_ , Frank Levy, and Richard J. Murnane, "The Skill Content of Recent Technological Change: An Empirical Exploration," *The Quarterly Journal of Economics*, November 2003, *118* (4), 1279–1333. 5

Becker, Joel, Nate Rush, Elizabeth Barnes, and David Rein, "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity," July 2025. 5

Bick, Alexander, Adam Blandin, and David Deming, "The Rapid Adoption of Generative AI," Technical Report w32966, National Bureau of Economic Research, Cambridge, MA September 2024. 6

Bloom, Nicholas, Luis Garicano, Raffaella Sadun, and John Van Reenen, "The Distinct Effects of Information Technology and Communication Technology on Firm Organization," *Management Science*, December 2014, *60* (12), 2859–2885. 5

Bresnahan, Timothy F. and Manuel Trajtenberg, "General Purpose Technologies 'Engines of Growth'?," *Journal of Econometrics*, January 1995, *65* (1), 83–108. 5

Brynjolfsson, Erik and Lorin M Hitt, "Beyond Computation: Information Technology, Organizational Transformation and Business Performance," *Journal of Economic Perspectives*, November 2000, *14* (4), 23–48. 5

_ , Bharat Chandar, and Ruyu Chen, "Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence," Technical Report, Working paper. 2025. 6

_ , Daniel Rock, and Chad Syverson, "The Productivity J-Curve: How Intangibles Complement General Purpose Technologies," *American Economic Journal: Macroeconomics*, January 2021, *13* (1), 333–372. 4, 23

_ , Danielle Li, and Lindsey Raymond, "Generative AI at Work," *The Quarterly Journal of Economics*, April 2025, *140* (2), 889–942. 6

Cameron, A Colin, Jonah B Gelbach, and Douglas L Miller, "Bootstrap-Based Improvements for Inference with Clustered Errors," *The review of economics and statistics*, 2008, *90* (3), 414–427. 52

Chatterji, Aaron, Thomas Cunningham, David Deming, Zoe Hitzig, Christopher Ong, Carl Yan Shan, and Kevin Wadman, "How People Use ChatGPT," Technical Report w34255, National Bureau of Economic Research, Cambridge, MA September 2025. 3, 6

Chen, Bei, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen, "CodeT: Code Generation with Generated Tests," November 2022. 9

Chen, Fiona and James Stratton, "Does AI Assistance Mean Engineers Are Doing Less Coding Work? Maybe Not," September 2025. 5

_ and _ , "Generative AI and Organizational Structure," *mimeo*, 2025. 5

Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba, "Evaluating Large Language Models Trained on Code," July 2021. 9

Choi, James, Dong Huang, Zhishu Yang, and Qi Zhang, "How Good Is AI at Twisting Arms? Experiments in Debt Collection," Technical Report w33669, National Bureau of Economic Research, Cambridge, MA April 2025. 6

Clark, A. and D. Chalmers, "The Extended Mind," *Analysis*, January 1998, *58* (1), 7–19. 5

Cui, Zheyuan Kevin, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz, "The Effects of Generative Ai on High Skilled Work: Evidence from Three Field Experiments with Software Developers," *Available at SSRN 4945566*, 2024. 5

Dell'Acqua, Fabrizio, Edward McFowland III, Ethan R Mollick, Hila Lifshitz-Assaf, Katherine Kellogg, Saran Rajendran, Lisa Krayer, Francois Candelon, and Karim R Lakhani, "Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality," *Harvard Business School Technology & Operations Mgt. Unit Working Paper*, 2023. 6

Dillon, Eleanor Wiske, Sonia Jaffe, Nicole Immorlica, and Christopher T. Stanton, "Shifting Work Patterns with Generative AI," July 2025. 6

Fischer, Alexander and David Roodman, "Fwildclusterboot: Fast Wild Cluster Bootstrap Inference for Linear Regression Models (Version 0.14.0)," 2021. 52

Forsgren, Nicole, Dustin Smith, Jez Humble, and Jessie Frazelle, "2019 Accelerate State of Devops Report," *DORA and Google Cloud, Tech. Rep*, 2019, *48455*. 8

_ , Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler, "The SPACE of Developer Productivity: There's More to It than You Think.," *Queue*, 2021, *19* (1), 20–48. 8

Fowler, Martin, "Continuous Integration," January 2024. 9

Friedman, Nat, "Introducing GitHub Copilot: Your AI Pair Programmer," June 2021. 1

Garicano, Luis and Esteban Rossi-Hansberg, "Organization and Inequality in a Knowledge Economy," *The Quarterly Journal of Economics*, November 2006, *121* (4), 1383–1435. 6

Gimbel, Martha, Molly Kinder, Joshua Kendall, and Maddie Lee, "Evaluating the Impact of AI on the Labor Market: Current State of Affairs," October 2025. 6

Hadfield, Gillian K. and Andrew Koh, "An Economy of AI Agents," September 2025. 6

Hampole, Menaka, Dimitris Papanikolaou, Lawrence D.W. Schmidt, and Bryan Seegmiller, "Artificial Intelligence and the Labor Market," Technical Report w33509, National Bureau of Economic Research, Cambridge, MA February 2025. 6

Handa, Kunal, Alex Tamkin, Miles McCain, Saffron Huang, Esin Durmus, Sarah Heck, Jared Mueller, Jerry Hong, Stuart Ritchie, Tim Belonax, Kevin K. Troy, Dario Amodei, Jared Kaplan, Jack Clark, and Deep Ganguli, "Which Economic Tasks Are Performed with AI? Evidence from Millions of Claude Conversations," February 2025. 6

Hindle, Abram, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu, "On the Naturalness of Software," *Communications of the ACM*, April 2016, *59* (5), 122–131. 8

Hoffmann, Manuel, Sam Boysel, Frank Nagle, Sida Peng, and Kevin Xu, "Generative AI and the Nature of Work," 2024. 5

Humlum, Anders and Emilie Vestergaard, "Large Language Models, Small Labor Market Effects," Technical Report w33777, National Bureau of Economic Research, Cambridge, MA May 2025. 3, 6

Hutchins, Edwin, *Cognition in the Wild* A Bradford Book, 8. pr ed., Cambridge, Mass.: MIT Press, 2006. 5

Ide, Enrique and Eduard Talamàs, "Artificial Intelligence in the Knowledge Economy," *Journal of Political Economy*, October 2025, pp. 000–000. 6

Jabarian, Brian and Luca Henkel, "Voice AI in Firms: A Natural Field Experiment on Automated Job Interviews," 2025. 6

Jahani, Eaman, Benjamin S. Manning, Joe Zhang, Hong-Yi TuYe, Mohammed Alsobay, Christos Nicolaides, Siddharth Suri, and David Holtz, "Prompt Adaptation as a Dynamic Complement in Generative AI Systems," April 2025. 6

Jimenez, Carlos E, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan, "SWE-bench: Can Language Models Resolve Real-World Github Issues?," in "The Twelfth International Conference on Learning Representations" 2024. 10

Katz, Lawrence F. and Kevin M. Murphy, "Changes in Relative Wages, 1963-1987: Supply and Demand Factors," *The Quarterly Journal of Economics*, February 1992, *107* (1), 35–78. 6

Kirsh, David and Paul Maglio, "On Distinguishing Epistemic from Pragmatic Action," *Cognitive Science*, October 1994, *18* (4), 513–549. 5

Kleinberg, Jon, Jens Ludwig, Sendhil Mullainathan, and Ziad Obermeyer, "Prediction Policy Problems," *American Economic Review*, May 2015, *105* (5), 491–495. 6

Korinek, Anton, "Generative AI for Economic Research: Use Cases and Implications for Economists," *Journal of Economic Literature*, December 2023, *61* (4), 1281–1317. 6

Le, Hung, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi, "CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning," November 2022. 9

Liang, Annie, "Artificial Intelligence Clones," April 2025. 6

Lichtinger, Guy and Seyed Mahdi Hosseini Maasoum, "Generative AI as Seniority-Biased Technological Change: Evidence from U.S. Résumé and Job Posting Data," 2025. 6

Liu, Xiao, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan

Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang, "AgentBench: Evaluating LLMs as Agents," in "The Twelfth International Conference on Learning Representations" 2024. 10

Mansfield, Edwin, "Technical Change and the Rate of Imitation," *Econometrica*, October 1961, *29* (4), 741. 5

Martin, Robert C., *Clean Code: A Handbook of Agile Software Craftsmanship* Robert C. Martin Series, Prentice Hall, 2012. 9

McElheran, Kristina, J. Frank Li, Erik Brynjolfsson, Zachary Kroff, Emin Dinlersoz, Lucia Foster, and Nikolas Zolas, "AI Adoption in America: Who, What, and Where," *Journal of Economics & Management Strategy*, March 2024, *33* (2), 375–415. 6

Noy, Shakked and Whitney Zhang, "Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence," *Science*, July 2023, *381* (6654), 187–192. 6

Paradis, Elise, Kate Grey, Quinn Madison, Daye Nam, Andrew Macvean, Vahid Meimand, Nan Zhang, Ben Ferrari-Church, and Satish Chandra, "How Much Does AI Impact Development Speed? An Enterprise-Based Randomized Controlled Trial," November 2024. 5

Peng, Sida, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer, "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot," February 2023. 5

Risko, Evan F and Sam J Gilbert, "Cognitive Offloading," *Trends in cognitive sciences*, 2016, *20* (9), 676–688. 5

Rothschild, David M., Markus Mobius, Jake M. Hofman, Eleanor W. Dillon, Daniel G. Goldstein, Nicole Immorlica, Sonia Jaffe, Brendan Lucier, Aleksandrs Slivkins, and Matthew Vogel, "The Agentic Economy," May 2025. 6

Russell, Stuart J. and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 4 ed., Pearson, 2020. 1

Sadowski, Caitlin, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli, "Modern Code Review: A Case Study at Google," in "Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice" ACM Gothenburg Sweden May 2018, pp. 181–190. 9

Schick, Timo, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds., *Advances in Neural Information Processing Systems*, Vol. 36 Curran Associates, Inc. 2023, pp. 68539–68551. 1

Shahidi, Peyman, Gili Rusak, Benjamin S Manning, Andrey Fradkin, and John J Horton, "The Coasean Singularity? Demand, Supply, and Market Design with AI Agents," *NBER Chapters*, 2025. 6

Shao, Yijia, Humishka Zope, Yucheng Jiang, Jiaxin Pei, David Nguyen, Erik Brynjolfsson, and Diyi Yang, "Future of Work with AI Agents: Auditing Automation and Augmentation Potential across the U.S. Workforce," June 2025. 6

Stevenson, Megan T. and Jennifer L. Doleac, "Algorithmic Risk Assessment in the Hands of Humans," *American Economic Journal: Economic Policy*, November 2024, *16* (4), 382–414. 6

Syverson, Chad, "What Determines Productivity?," *Journal of Economic Literature*, June 2011, *49* (2), 326–365. 5

Tomlinson, Kiran, Sonia Jaffe, Will Wang, Scott Counts, and Siddharth Suri, "Working with AI: Measuring the Applicability of Generative AI to Occupations," September 2025. 6

Vafa, Keyon, Ashesh Rambachan, and Sendhil Mullainathan, "Do Large Language Models Perform the Way People Expect? Measuring the Human Generalization Function," in "International Conference on Machine Learning" PMLR 2024, pp. 48919–48937. 6

_ **, Sarah Bentley, Jon Kleinberg, and Sendhil Mullainathan**, "What's Producible May Not Be Reachable: Measuring the Steerability of Generative Models," March 2025. 6

Wooldridge, Michael and Nicholas R. Jennings, "Intelligent Agents: Theory and Practice," *The Knowledge Engineering Review*, June 1995, *10* (2), 115–152. 1

Yang, John, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press, "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," November 2024. 10

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao, "React: Synergizing Reasoning and Acting in Language Models," in "International Conference on Learning Representations (ICLR)" 2023. 1

Zhang, Jiajie and Donald A. Norman, "Representations in Distributed Cognitive Tasks," *Cognitive Science*, January 1994, *18* (1), 87–122. 5, 27

Zhao, Wenting, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng, "WildChat: 1M ChatGPT Interaction Logs in the Wild," May 2024. 6

A. Appendix

Appendix A.1 includes details on labeling work history and message intent. Appendix A.2 includes additional analysis related to the main results.

A.1. Labeling Details

User characteristics Role, seniority, and years of work experience of sampled users were determined by a labeling prompt combined with in-context examples. Sector is an internal Anysphere designation at the organization level. De-identified data on work history summaries was constructed by Anysphere by matching a subset of users to publicly available profile information. Each de-identified history summary used for labeling contained a list of job titles and year ranges. The 100 most common job titles were manually labeled with role and seniority, and passed as examples alongside a prompt detailed in Figure A1 to label the history summaries. The labeling was conducted using an internal inference endpoint.

Intent labeling Message intent was determined by a labeling prompt detailed in Figure A2. Message intent was inferred for users on individual subscriptions who had opted into sharing message language. The labeling was conducted using an internal inference endpoint using batches of 16 messages. In the main text, a `null` label is referred to as "Other."

Figure A1: Characteristics prompt.

User Characteristics From Work History

Given a JSON array of work history, extract the following fields:

- earliest_professional_employment_start_year: the chronologically first professional employment start year as a 4-digit integer; ignore any student/intern/volunteer/teaching/campus/research assistant roles.
- role: choose ONE from this set exactly: {role_options}.
- seniority: choose ONE from this set exactly: {seniority_options}.

Classification guidelines:

- Weigh the current/most recent title most strongly when determining role.
- If the role is general management, people management, operations, or founder, and does not clearly map to software, product, design, data, marketing, or sales management, set role to 'business/operations'.

Seniority guidelines:

- Set seniority='executive/founder' ONLY for explicit executive signals: C-level (e.g., CEO, CTO, CFO, COO, CIO, CISO ...), President, Vice President/VP/SVP/EVP, Founder/Co-Founder, or titles containing 'Global Head' or 'Worldwide Head'.
- Otherwise, classify titles containing 'Director', 'Senior Director', 'Executive Director', 'Head of', or 'Manager' as seniority='management' (not 'executive/founder').
- management: People management, including Director, Senior Director, Executive Director, Head of, Manager, Senior Manager, Principal Manager, Group Manager (not 'executive/founder').
- staff/principal: Staff, Principal, Distinguished, Fellow, Architect, Solutions/Software Architect. Numeric levels IV/V (or 4/5) map here when IC roles.
- senior: Senior, Lead (when not 'Team Lead' and not paired with Manager/Director/Head), IC level III (or 3).
- junior: Software Engineer, Product Manager, Associate, Junior, IC level I/II (or 1/2). Default for plain 'Engineer/Developer' without modifiers.
- Conflicts: management overrides senior/staff terms in the same title (e.g., 'Senior Director' => management).

Examples of labeled data (use these as guidance):
{examples_text}
Return ONLY valid JSON with keys: earliest_professional_employment_start_year, role, seniority. Do not include comments or markdown fences.

Figure A2: Intent prompt

Intent From Message Content

SCHEMA (output must conform exactly):

```
{
 "items": [
   {
    "intent": string | null // one of the allowed INTENT values, or null
   }
 ]
}
```

ALLOWED VALUES:

- INTENT: implement | plan | explain code/codebase | explain error
  - implement: user requests code changes, creation of artifacts, or execution-oriented changes (scripts, configs, migrations).
  - plan: user requests approach/architecture/steps without writing code yet.
  - explain code/codebase: user asks how existing code/repo works or where something is implemented.
  - explain error: user asks to diagnose a failure using logs/tracebacks/errors or observed malfunction.

DECISION RULES (precedence and tie-breakers):

- Choose exactly ONE intent:
  1. If the user asks for concrete changes or new artifacts (code/CI/build/scripts/config) now or within the same request → intent = implement. This dominates over other intents.
  2. Else if the primary request is to diagnose a failure and includes logs/tracebacks/observed malfunction and no action is requested → intent = explain error.
  3. Else if the user asks how the existing system/code works and no action is requested → intent = explain code/codebase.
  4. Else if the user asks for an approach or planning without executing changes → intent = plan.
- Purity constraint: choose plan/explain only when there is NO imperative to change, run, fix, or create.
- Note: When analysis and action both appear (e.g., "investigate then fix", "write a plan and implement it"), choose implement.

Additional tie-breakers and lexical cues:

- If a request contains even a single directive to make changes now OR as part of the same ask (e.g., "make a plan, then proceed", "investigate then fix", "write the plan and implement it"), choose implement.
- Authoring non-code artifacts (docs, posts, summaries, test cases) counts as implement.
- If logs/tracebacks are provided AND the user asks to fix/apply a change, choose implement (not explain error).
- Requests to verify what exists in this repo (e.g., "check routes", "is X implemented?", "where is Y?") without asking for changes → explain code/codebase.
- Requests that report ongoing/frequent failures or malfunctions without prescribing changes (e.g., "something is wrong", "this keeps happening", errors/tracebacks) → explain error.
- Requests to run tests/build/deploy or operate tools/environments/CI/CD (e.g., "run all tests", "build and run the app", "set up staging/prod") → implement.
- Large rename/refactor across the whole project (e.g., rename categories everywhere) → implement.
- "How can I ..." questions that clearly expect concrete code changes or command execution in this repo (e.g., "adjust method X", "use package Y to compute Z in code") → implement (not plan).
- Don't infer missing details. If a field is uncertain, set it to null.

EXAMPLES (for calibration, not to be output):

- "Why does this stack trace fail? [traceback. . .]" → intent: explain error
- "Replace navbar with shadcn menu" → intent: implement
- "How do we assign proxies when creating phones?" → intent: explain code/codebase

- "Use gh CLI to pull PR #225 and address comments" → intent: implement
- "Plan the approach for migrating X" → intent: plan
- "Tests are failing with this output: [test failure logs]" → intent: explain error
- "Refactor large files into modules and add docs" → intent: implement
- "What does the authentication middleware do?" → intent: explain code/codebase
- "Let's check the file upload pipeline and confirm routes are implemented" → intent: explain code/codebase
- "Can you add logging to the auth module?" → intent: implement
- "Something is wrong in this implementation; help identify the problem" → intent: explain error
- "Should I use Redis or Memcached for caching here?" → intent: plan
- "Run all tests in parallel with run_all_test.sh and update report" → intent: implement
- "Walk me through the payment processing flow." → intent: explain code/codebase
- "Rename component categories across the entire project (elements→primitives, etc.)" → intent: implement
- "The build breaks with this message: [build error]" → intent: explain error
- "Use this function to sum and format amounts in code?" → intent: implement
- "Where are environment variables configured?" → intent: explain code/codebase
- "Draft a test plan for feature X; do not implement yet." → intent: plan
- "Adjust getLocalIP to only return addresses starting with '10.'" → intent: implement
- "API returns 403 but I'm authenticated; what's happening?" → intent: explain error
- "Could you refactor this to use async/await?" → intent: implement
- "What's the difference between ServiceA and ServiceB?" → intent: explain code/codebase
- "Scan the repo for modern frontend components and stage them in a folder" → intent: implement
- "Memory usage spikes to 8GB; help diagnose the leak." → intent: explain error
- "What would be the best approach to migrate from REST to GraphQL?" → intent: plan
- "Build and run the repo, then test connecting to providers" → intent: implement
- "Explain how module Y resolves dependencies; don't change anything" → intent: explain code/codebase
- "Would you mind updating the dependencies?" → intent: implement
- "Diagnose why this 500 happens and do not fix yet" → intent: explain error
- "Investigate this stack trace and fix the bug" → intent: implement
- "Outline a blue/green rollout and rollback plan for service X." → intent: plan
- "Help me set up the development environment." → intent: implement
- "In src/auth/AuthService.ts, implement token refresh retry with jitter." → intent: implement
- "Compare authentication strategies for this app; don't implement." → intent: plan
- "Given this traceback for migration M, apply the fix and re-run." → intent: implement
- "Write unit tests for the UserService class." → intent: implement
- "Step 1: install deps; Step 2: run scripts/seed.sh; Step 3: verify counts." → intent: implement
- "Document the API endpoints in OpenAPI format." → intent: implement
- "Change LOG_LEVEL=warning in docker-compose.yml." → intent: implement
- "The /users page renders in ~5s; reduce to under 1s." → intent: implement
- "Optimize this slow database query." → intent: implement
- "Create models/stg_orders.sql and register dags/dbt_refresh.py in Airflow." → intent: implement
- "Train a PyTorch classifier and log metrics to MLflow; add inference/predict.py." → intent: implement

STRICT RULES:
- Choose exactly ONE value for intent when evidence exists; otherwise set to null.
- Do not guess. If uncertain, set intent to null.
- Apply the precedence rules above for intent.
- Output ONLY the JSON object adhering to the schema above. No markdown, comments, or additional keys.

Table A3: Summary statistics from work history classification.

	Full Sample	Agent Metrics	Tab Metrics
N users	86,665	44,160	40,533
N organizations	1,000	997	991
Role:			
Software	65,864 (76.0%)	35,457 (80.3%)	32,988 (81.4%)
Data/ML	7,862 (9.1%)	3,731 (8.4%)	3,842 (9.5%)
Business/Operations	6,089 (7.0%)	2,546 (5.8%)	1,990 (4.9%)
Product	3,222 (3.7%)	1,134 (2.6%)	791 (2.0%)
Design	1,921 (2.2%)	710 (1.6%)	444 (1.1%)
Sales	900 (1.0%)	304 (0.7%)	262 (0.6%)
Marketing	807 (0.9%)	278 (0.6%)	216 (0.5%)
Sector:			
Software & Developer Tools	34,222 (39.5%)	17,693 (40.1%)	16,636 (41.0%)
Finance & Fintech	18,178 (21.0%)	9,297 (21.1%)	8,590 (21.2%)
Consumer & Retail	11,871 (13.7%)	5,304 (12.0%)	4,672 (11.5%)
Media & Advertising	7,673 (8.9%)	3,975 (9.0%)	3,528 (8.7%)
Logistics & Platforms	6,455 (7.4%)	3,451 (7.8%)	3,185 (7.9%)
Healthcare & Life Sciences	4,538 (5.2%)	2,442 (5.5%)	2,192 (5.4%)
Consulting & Professional Services	3,728 (4.3%)	1,998 (4.5%)	1,730 (4.3%)
Seniority:			
Junior	34,290 (39.6%)	19,483 (44.1%)	18,231 (45.0%)
Senior	26,640 (30.7%)	13,926 (31.5%)	12,846 (31.7%)
Management	12,956 (14.9%)	4,874 (11.0%)	3,932 (9.7%)
Staff/Principal	9,590 (11.1%)	4,535 (10.3%)	4,385 (10.8%)
Executive/Founder	3,189 (3.7%)	1,342 (3.0%)	1,139 (2.8%)
Years of experience:			
Mean	11.7	10.9	10.8
Standard deviation	6.7	6.6	6.4
25th percentile	7	6	6
Median	11	10	10
75th percentile	16	15	14

Notes – This table reports summary statistics of user characteristics variables on the full sample of organization users. It also reports summary statistics on subsamples with agent edits or tab autocompletion accepts observed during the week of July 7, 2025.

A.2. Additional Exhibits

I include additional exhibits related to the analysis.

- Table A4 and Table A5 report results of regressions of relative agent accept rates on work experience.
- Figure A6 reports relative autocompletion accept rates by user characteristics.
- Figure A7 reports dynamic difference-in-differences estimates for additional outcomes.
- Table A8 reports pre-period balance statistics for the merge metadata sample.
- Figure A9 plots raw outcomes across the groups.
- Table A10 reports period-by-period means for the eligible and baseline groups.
- Table A11 reports results using the wild cluster bootstrap-t.
- Table A12 reports additional estimates related to lines edited per merge and number of files touched per merge.
- Table A13 reports subsample analyses by partitions of the eligible group.
- Table A14 reports results of regressions of message intent rates on work experience and total usage.

Experience gap in agent accept rates I estimate regressions following

$$\text{Accept Rate}_i = \beta \text{Work Experience}_i + \delta_{c(i)} + \varepsilon_i$$

for user i, where $\delta_c(i)$ corresponds to fixed effects for characteristics c. Table A4 reports the results of these regressions.

Table A4: Work experience and agent accept rates.

	Relative Accept Rate $\times 100$			
	(1)	(2)	(3)	(4)
Work Experience (σ)	5.97	5.82	5.85	4.87
	(0.20)	(0.20)	(0.20)	(0.23)
Constant	100.00			
	(0.20)			
R^2	0.019	0.021	0.022	0.025
Within R^2		0.018	0.018	0.010
Observations	44,160	44,160	44,160	44,160
Role FE		✓	✓	✓
Sector FE			✓	✓
Seniority FE				✓

Notes – This table reports results of regressions of agent accept rates on work experience. The second column adds role fixed effects, the third adds sector fixed effects, and the fourth adds seniority fixed effects. One standard deviation in work experience corresponds to 6.6 years of experience. Robust standard errors are reported in parentheses.

Experience gap in agent accept rates, adjusting for total usage I estimate regressions following

$$\text{Accept Rate}_i = \beta_1 \text{Work Experience}_i + \beta_2 \text{Total Usage}_i + \delta_{c(i)} + \varepsilon_i$$

for user i, where $\delta_c(i)$ corresponds to fixed effects for characteristics c. Table A5 reports the results of these regressions.

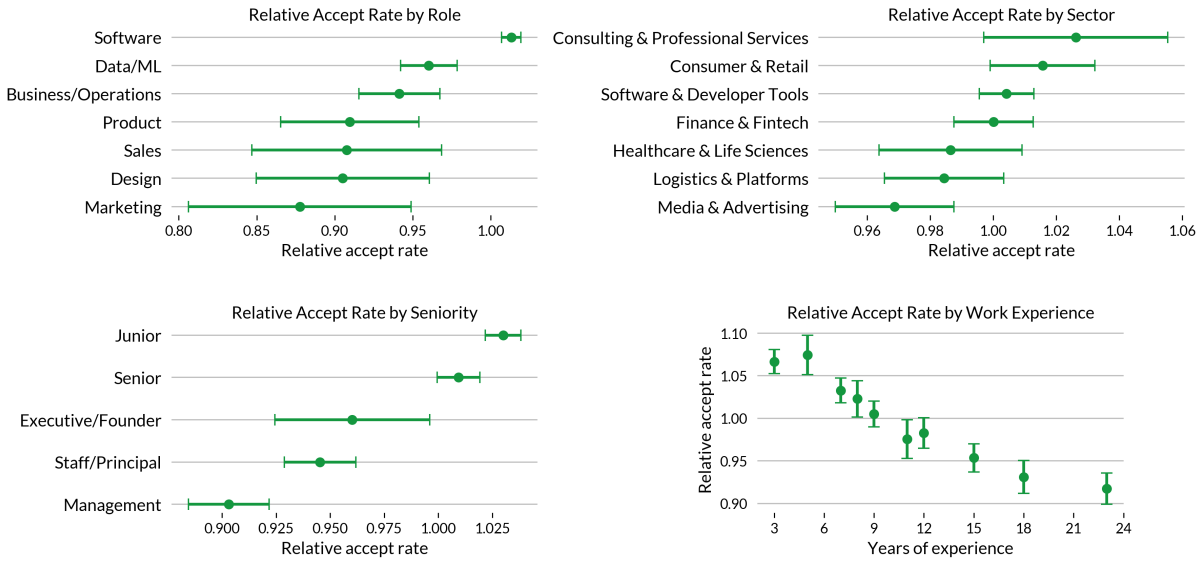
Table A5: Work experience and agent accept rates, adjusting for total usage.

	Relative Accept Rate $\times 100$			
	(1)	(2)	(3)	(4)
Work Experience (σ)	5.96	5.82	5.85	4.85
	(0.20)	(0.20)	(0.20)	(0.23)
Total Usage (σ)	1.60	1.53	1.48	1.54
	(0.17)	(0.17)	(0.17)	(0.17)
Constant	100.00			
	(0.20)			
R^2	0.021	0.023	0.023	0.026
Within R^2		0.019	0.019	0.011
Observations	44,160	44,160	44,160	44,160
Role FE		✓	✓	✓
Sector FE			✓	✓
Seniority FE				✓

Notes – This table reports results of regressions of agent accept rates on work experience and total usage. The second column adds role fixed effects, the third adds sector fixed effects, and the fourth adds seniority fixed effects. One standard deviation in work experience corresponds to 6.6 years of experience. One standard deviation in total usage corresponds to 76.6 messages per week. Robust standard errors are reported in parentheses.

Autocompletion accept rates I match the organization user sample to 40,533 users with at least one accepted autocompletion from the week of July 7, 2025. Figure A6 reports autocompletion accept rates by user characteristics.

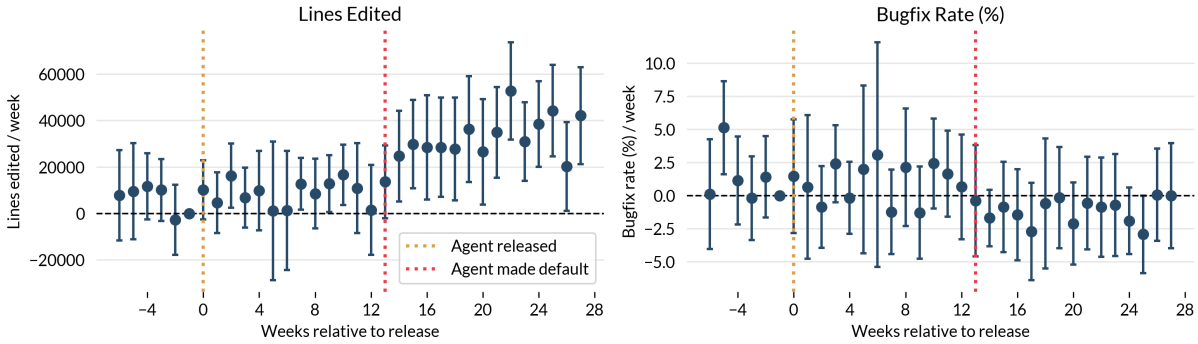
Figure A6: Autocompletion accept rates by user characteristics.



Notes – This figure plots autocompletion accept rates across role, sector, seniority, and decile of work experience. Error bars report 95% confidence intervals for group means.

Additional dynamic difference-in-differences plots Figure A7 reports dynamic difference-in-differences estimates for the additional outcome variables.

Figure A7: Estimated effects of the agent feature on additional outcomes.



Notes – This figure plots dynamic difference-in-differences estimates of lines edited and bugfix rates among organizations eligible for the agent feature. The estimates follow the specification in Equation (1). Vertical lines denote the weeks of the release date and default date of the agent feature. Standard errors are clustered by organization.

Balance table Table A8 is a balance table for the difference-in-differences specification.

Table A8: Balance table.

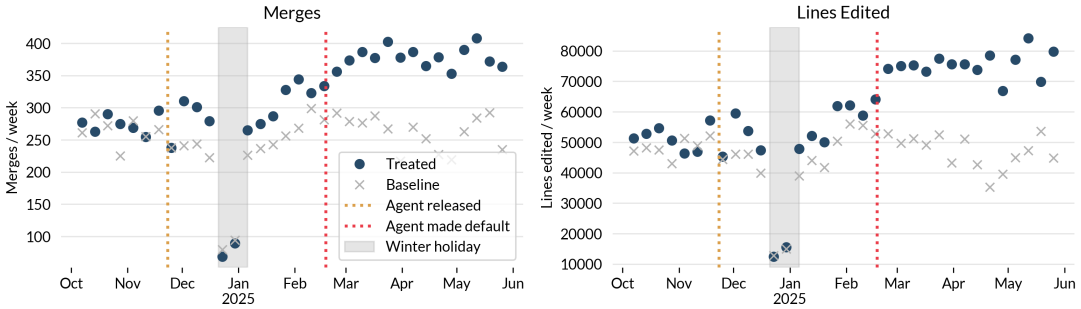
	Eligible Group	Baseline Group	Normalized Difference
Merges	271.7	264.3	0.04
Lines Edited (K)	50.5	47.7	0.07
Bugfix Rate (%)	10.4	10.0	0.06
Revert Rate (%)	1.07	0.87	0.20
Lines Edited / Merge	197.0	195.9	0.02
Files Touched / Merge	6.0	6.1	-0.11
Organizations	24	8	

Notes – This table reports pre-agent release means of the main outcome measures for the eligible and baseline groups. The eligible group contains organizations that adopted the Cursor platform before the sample period began in October 2024. The baseline group contains organizations that did not use the Cursor platform during the sample period. Note that the average of lines edited / merge (a mean of ratios) is greater than average lines edited / average merges (a ratio of means) since weekly merges and lines edited per merge are negatively correlated.

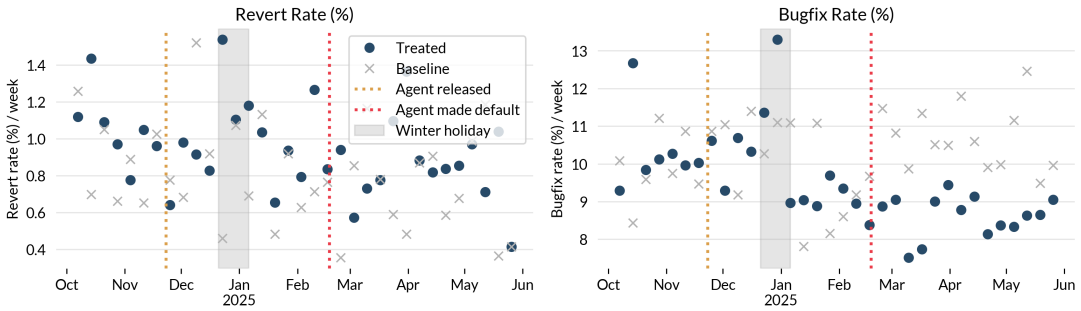
Raw outcomes Figure A9 plots mean outcomes across the groups. There is seasonality in merges and lines edited during holidays, which is adjusted for in the results that use the difference-in-differences specification.

Figure A9: Outcomes surrounding the agent feature release.

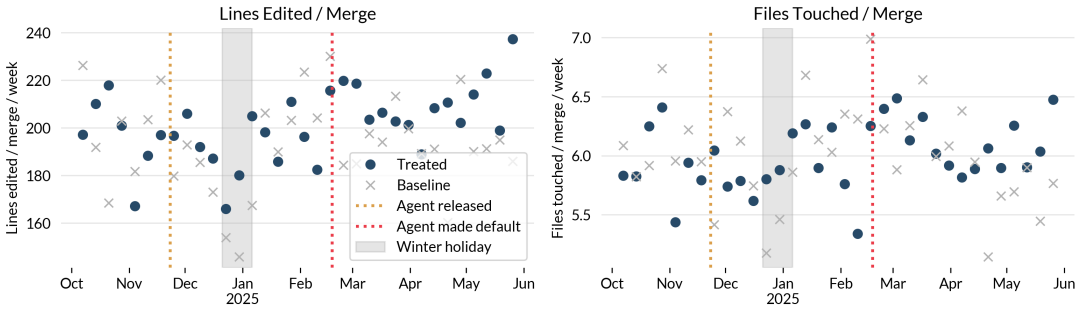
(a) Output (larger outcomes indicate higher output).



(b) Quality (smaller outcomes indicate lower rates of short-run issues).



(c) Scope (larger outcomes proxy for larger-scoped merges).



Notes – This figure plots mean outcomes related to throughput (merges and lines edited), quality (bugfix rate and revert rate), and scope (lines edited per merge and files touched per merge) for organizations in the eligible and baseline groups. Vertical lines denote the release date and default date of the agent feature.

Outcome table Table A10 reports means of the output and quality measures across release periods and organization groups.

Table A10: Mean outcomes across groups and periods.

Period	Group	Merges	Lines Edited (K)	Revert Rate (%)	Bugfix Rate (%)
Pre Release	Eligible	271.7	50.5	1.07	10.4
	Baseline	264.3	47.7	0.87	10.0
Post Release (Full)	Eligible	322.7	62.4	0.92	9.3
	Baseline	245.2	44.8	0.79	10.3
Post Release & Pre Default	Eligible	262.1	48.1	0.99	10.0
	Baseline	224.5	41.8	0.85	9.9
Post Release & Post Default	Eligible	375.2	74.8	0.86	8.6
	Baseline	263.1	47.4	0.73	10.6

Notes – This table reports mean outcomes across periods for the eligible and baseline groups.

Bootstrap Table A11 reports estimates from the main difference-in-differences specification. Inference uses the wild cluster bootstrap-t (Cameron et al., 2008), clustered by organization and implemented via the `fwildclusterboot` package (Fischer and Roodman, 2021). The analysis is computed using 9,999 replications with Rademacher weights, absorbing week fixed effects. Two-sided p-values and 95% confidence intervals are reported. The original t-statistic from clustering by organization is also reported. There are 24 eligible organizations and 8 baseline organizations.

Table A11: Results from bootstrap inference.

(a) Post-release (full) versus pre-release.

Outcome	Estimate	p-value	CI Low	CI High	t-statistic
Merges	70.09	0.002	31.74	108.63	3.77
Lines Edited (K)	14.80	0.006	4.68	24.70	3.01
Revert Rate (%)	-0.074	0.595	-0.361	0.212	-0.54
Bugfix Rate (%)	-1.41	0.030	-2.66	-0.16	-2.42

(b) Post-default versus pre-release.

Outcome	Estimate	p-value	CI Low	CI High	t-statistic
Merges	104.69	0.002	52.99	156.36	4.10
Lines Edited (K)	24.61	0.003	9.36	39.85	3.29
Revert Rate (%)	-0.081	0.624	-0.429	0.274	-0.49
Bugfix Rate (%)	-2.40	0.004	-3.85	-0.92	-3.43

Notes – These tables report the results of the main difference-in-differences regressions, along with wild cluster bootstrap p-values and confidence intervals.

Merge scope Table A12 reports the results of regressions following

$$Y_{i,t} = \alpha_i + \alpha_t + \beta_1 \times \text{Eligible}_i \times \text{Post Release}_t + \beta_2 \times \text{Eligible}_i \times \text{Post Default}_t + \varepsilon_{i,t} \quad (5)$$

These regressions present subperiod analyses that divide the post-release period into a pre-default and post-default period.

The regressions include outcome variables related to scope: Lines edited per merge and files touched per merge. Analogously to lines edited, number of files touched per merge is winsorized at the 1% level at the right tail, and then averaged to the weekly level. The merge scope subsample analysis considers changes in small and large merges, defined as those below or above median lines edited or files touched for the eligible group in the pre-release period.

Table A12a does not find evidence of changes in lines edited per merge or files touched per merge. Table A12b indicates that there were increases in both small and large merges. In the eligible group, pre-release means of merges per week were 137.0 for small merges and 134.5 for large merges (by lines edited), and 146.1 for small merges and 125.6 for large merges (by files touched).

In the initial period of agent use, there is directional evidence that eligible organizations produced more merges of both types. Changes are computed by dividing each $\hat{\beta}_1$ by pre-release means. When scope is defined by lines edited, eligible organizations produced small merges 12% more often and large merges 10% more often. When scope is defined by files touched, eligible organizations produced small merges 10% more often and large merges 12% more often.

In the default period, eligible organizations produced more merges of both types. Changes are computed by dividing each $\hat{\beta}_1 + \hat{\beta}_2$ by pre-release means. When scope is defined by lines edited, eligible organizations produced small merges 33% more often and large merges 45% more often. When scope is defined by files touched, eligible organizations produced small merges 39% more often and large merges 38% more often.

Table A12: Estimates from subperiod and subsample analyses.

(a) Estimates from subperiod specification across outcomes.

	Merges (1)	Lines Edited (K) (2)	Revert Rate (%) (3)	Bugfix Rate (%) (4)	Lines / Merge (5)	Files / Merge (6)
Eligible × Post Release	30.17	3.47	-0.07	-0.27	3.29	0.08
	(16.89)	(3.33)	(0.15)	(0.69)	(20.85)	(0.43)
Eligible × Post Default	74.52	21.14	-0.02	-2.12	10.48	0.22
	(23.37)	(6.63)	(0.15)	(0.75)	(12.48)	(0.26)
R^2	0.87	0.83	0.27	0.53	0.50	0.50
Within R^2	0.04	0.04	0.00	0.01	0.00	0.00
Observations	1,088	1,088	1,088	1,088	1,088	1,088
Organization FE	✓	✓	✓	✓	✓	✓
Time FE	✓	✓	✓	✓	✓	✓

(b) Estimates from subperiod specification across merge scope subsamples.

	Merges			
	Small (Lines) (1)	Large (Lines) (2)	Small (Files) (3)	Large (Files) (4)
Eligible × Post Release	16.84	13.38	14.95	15.22
	(8.46)	(9.99)	(8.30)	(10.64)
Eligible × Post Default	27.82	46.31	41.70	32.82
	(9.06)	(15.79)	(15.37)	(10.22)
R^2	0.88	0.86	0.86	0.87
Within R^2	0.03	0.04	0.03	0.04
Observations	1,088	1,088	1,088	1,088
Organization FE	✓	✓	✓	✓
Time FE	✓	✓	✓	✓

Notes – These tables report difference-in-differences estimates of outcomes among organizations eligible for the agent feature. The estimates follow the specification in Equation (5). The top panel reports estimates related to output, quality, and scope. The bottom panel compares estimates filtering to merges above and below the median of lines edited or files touched. Standard errors clustered by organization are reported in parentheses.

Effects by organization subsamples Table A13 reports variations of the main difference-in-differences results that split the eligible group across pre-release outcomes. These outcomes are weekly merges, weekly lines edited, revert rates, and bugfix rates. Eligible organizations are ranked by pre-release means of each outcome, and then split along the median. The results indicate that eligible organizations with higher pre-release revert rates had no higher increases in merges after the agent was released than eligible organizations with lower pre-release revert rates.

Table A13: Subsample analysis by pre-release merge characteristics.

(a) Post-release (full) versus pre-release.

Outcome for Subsample Split:	Merges		Lines Edited		Revert Rate		Bugfix Rate	
Eligible Group Subsample:	Low	High	Low	High	Low	High	Low	High
Dependent Variable:	Merges							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Eligible × Post Release	45.45	94.74	62.73	77.45	73.94	66.25	72.00	68.19
	(13.94)	(30.52)	(18.34)	(29.86)	(25.41)	(24.28)	(27.02)	(22.52)
R^2	0.88	0.84	0.88	0.85	0.88	0.86	0.85	0.90
Within R^2	0.03	0.03	0.04	0.02	0.02	0.03	0.02	0.03
Observations	680	680	680	680	680	680	680	680
Organization FE	✓	✓	✓	✓	✓	✓	✓	✓
Time FE	✓	✓	✓	✓	✓	✓	✓	✓
Pre-release mean merges:	101.29	442.17	122.60	420.86	261.50	281.96	320.29	223.17
Percent change:	44.9%	21.4%	51.2%	18.4%	28.3%	23.5%	22.5%	30.6%

(b) Post-default versus pre-release.

Outcome for Subsample Split:	Merges		Lines Edited		Revert Rate		Bugfix Rate	
Eligible Group Subsample:	Low	High	Low	High	Low	High	Low	High
Dependent Variable:	Merges							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Eligible × Post Default	51.05	158.33	74.63	134.76	113.01	96.38	113.03	96.36
	(17.55)	(41.32)	(24.93)	(41.94)	(39.74)	(30.90)	(35.52)	(35.67)
R^2	0.90	0.88	0.90	0.88	0.91	0.89	0.88	0.92
Within R^2	0.05	0.12	0.07	0.09	0.08	0.08	0.07	0.09
Observations	420	420	420	420	420	420	420	420
Organization FE	✓	✓	✓	✓	✓	✓	✓	✓
Time FE	✓	✓	✓	✓	✓	✓	✓	✓
Pre-release mean merges:	101.29	442.17	122.60	420.86	261.50	281.96	320.29	223.17
Percent change:	50.4%	35.8%	60.9%	32.0%	43.2%	34.2%	35.3%	43.2%

Notes – This table reports difference-in-differences estimates of merges among subsamples of organizations eligible for the agent feature. In each regression, the eligible group is split across medians of pre-release outcome variables. The estimates follow the specification in Equation (2). Standard errors clustered by organization are reported in parentheses.

Message intent frequency by work experience and usage Table A14 reports estimates of the relationship between message intent frequency, work experience, and usage.

Table A14: Message intent by experience and usage

	Implement (1)	Plan (2)	Explain Code (3)	Explain Error (4)	Other (5)
Work Experience (σ)	-0.60	1.58	-0.87	-0.22	0.11
	(0.61)	(0.40)	(0.35)	(0.24)	(0.09)
Total Messages (σ)	1.81	-1.23	-1.07	0.60	-0.11
	(0.48)	(0.30)	(0.27)	(0.24)	(0.08)
Constant	58.91	15.02	13.20	11.23	1.64
	(0.58)	(0.37)	(0.34)	(0.27)	(0.10)
R^2	0.01	0.03	0.02	0.01	0.00
Observations	1,000	1,000	1,000	1,000	1,000

Notes – This table reports regression estimates of message intent frequency on work experience and usage. One standard deviation in work experience corresponds to 8.4 years of experience. One standard deviation in total messages corresponds to 83.3 conversation-starting messages over the two-month sample period. Robust standard errors are reported in parentheses.

Message intent and agent accept rates How do agent accept rates relate to message intent styles? I relate initial message intent frequency to accept rates at the user level. I use the relative agent accept rate from the week of July 7, 2025, during which edit outcome data is available. This match yields a sample of 685 users, with a slight increase in the standard deviation of work experience from 8.4 to 8.6. Although the intent rate and accept rate samples are temporally disjoint, there are no institutional details that would suggest large distribution shifts in the one week in between the samples. Table A15a shows that users who send more planning messages to start conversations also have higher agent accept rates. Table A15b shows that both work experience and plan rates are positively associated with agent accept rates.

Table A15: Planning rates and accept rates.

(a) Accept rate by initial message intent rate.

	Relative Accept Rate $\times 100$				
	(1)	(2)	(3)	(4)	(5)
Implement Rate (σ)	-1.95				
	(1.27)				
Plan Rate (σ)		4.35			
		(1.24)			
Explain Code/Codebase Rate (σ)			-1.45		
			(1.42)		
Explain Error Rate (σ)				-0.49	
				(1.22)	
Other Rate (σ)					1.96
					(1.48)
Constant	100.00	100.00	100.00	100.00	100.00
	(1.31)	(1.30)	(1.31)	(1.31)	(1.31)
R^2	0.003	0.016	0.002	0.000	0.003
Observations	685	685	685	685	685

(b) Accept rate by plan rate and work experience.

	Relative Accept Rate $\times 100$		
	(1)	(2)	(3)
Work Experience (σ)	7.89		7.40
	(1.17)		(1.19)
Plan Rate (σ)		4.35	3.21
		(1.24)	(1.24)
Constant	100.00	100.00	100.00
	(1.27)	(1.30)	(1.27)
R^2	0.053	0.016	0.062
Observations	685	685	685

Notes – These tables report the results of cross-sectional regressions of accept rates on initial message intent rates. The standard deviations of the implement, plan, explain code/codebase, explain error, and other intent rates are 18.6, 11.8, 11.1, 8.5, and 2.6, respectively. One standard deviation in work experience corresponds to 8.6 years of experience. Robust standard errors are reported in parentheses.