The Impact of Generative AI on Productivity: Automation, Coordination, and Reallocation in Workplace Software Development

(Authors' names blinded for peer review)

Manuscript Version: March 9, 2026

The rapid adoption of generative AI in workplace has raised critical questions about its impact on employee productivity, work time and skill heterogeneity. To address these questions, we collaborate with a major technology company and leverage the deployment of an AI coder (i.e., AI coding assistant) as our empirical setting. Conceptualizing workplace software development as an interdependent three-layer workflow—production, integration, and communication, we employ a regression discontinuity in time design on granular daily behavioral data from 200 developers over four months. We find that the AI coder increases coding productivity by 6.9-8.4% and decreases working hours by 3.1-4.3%, without compromising code quality. The productivity effects exhibit significant heterogeneity across skill levels: lower-skilled developers experience larger productivity gains. We identify three mechanisms underlying the productivity gains: (1) at the production layer, the AI coder substitutes human effort through the automation of routine tasks; (2) at the integration layer, it decreases peer review time, significantly reducing coordination costs; and (3) at the communication layer, the time capacity saved from production and integration drives a resource reallocation toward collaborative problem-solving, evidenced by more frequent and longer technical meetings. In addition, the AI coder also enhances developers' capacity in complex problem-solving where higher-skilled developers demonstrate larger improvements, suggesting that while the AI coder serves as an equalizer for general coding work, it may amplify skill advantages in technically demanding work. Our findings contribute to understanding how AI reshapes employee productivity and time allocation in the context of workplace software development.

Key words: Generative AI, Productivity, Skill Heterogeneity, Work Time, Communication

1. Introduction

Generative artificial intelligence (GenAI) is fundamentally transforming the modern workplace. According to McKinsey research (Chui et al. 2023), GenAI has the potential to automate work activities that absorb 60-70% of employees' time today, thereby augmenting the capabilities of individual workers. Moreover, this report estimates that it will automate half of today's work activities between 2030 and 2060, and boost productivity growth by 0.5-3.4 percentage points annually. Recent academic evidence supports these projections, demonstrating AI-induced productivity gains in various fields, such as customer service, transportation, and professional writing (e.g., Brynjolfsson et al. 2025, Kanazawa et al. 2025, Noy and Zhang 2023, Zhang and Narayandas 2025).

Naturally, the information services sector has emerged as the frontrunner in this GenAI-induced transformation, demonstrating both the largest share of work time spent using GenAI and the highest time savings (Bick et al. 2025). One of the most significant areas within information services sector is software development, a typical knowledge-intensive domain. With the advent of general AI tools (e.g., ChatGPT) and AI coding assistants (hereinafter referred to as AI coder, such as Copilot and Cursor), software development has experienced profound changes in how developers write, review, and deploy code, reshaping programming productivity. Understanding how AI coders impact productivity in workplace settings is therefore crucial for organizations seeking to harness this technology effectively.

Previous studies on the productivity effects of GenAI have primarily focused on general knowledge work and service contexts (Tong et al. 2021, Luo et al. 2021, Brynjolfsson et al. 2025, Noy and Zhang 2023). Recent research extends this literature stream to software development, recognizing its unique characteristics and potential of AI coders (Cui et al. 2026, Song et al. 2024). Software development is a highly knowledge-intensive process, where each decision builds on existing codebases and requires deep technical expertise. Moreover, software development involves mandatory peer reviews and intensive coordination, distinguishing it from the individual tasks studied previously (e.g., Noy and Zhang 2023, Brynjolfsson et al. 2025). Within the software development

domain, however, existing research has focused on open-source environments (e.g., Song et al. 2024, Yeverechyahu et al. 2025, Li et al. 2025), leaving AI coders in workplace understudied. This gap is significant because workplace software development operates within different organization structures and collaboration processes compared to open-source settings.

To understand the effects of AI coders in the workplace setting, we conceptualize workplace software development as a three-layer workflow: production (code generation), integration (peer review), and communication (collaborative meetings). This framework reveals why workplace contexts present distinct theoretical challenges compared to open-source setting. First, workplace developers[1] work under resource constraints where time allocated to individual coding necessarily limits capacity for review and communication activities. Second, workplace settings involve intensive cross-functional communication between developers, product managers, and organizational stakeholders to align technical implementation with business objectives. Third, these activities form an interdependent system where bottlenecks in any layer can constrain overall productivity, unlike the more flexible, individually-driven work in open-source environments. Despite the significance of understanding how AI affects this coupled system, research examining AI coders in workplace settings remains limited, with Cui et al. (2026) as a recent exception documenting productivity effects. However, the underlying mechanisms through which AI affects each layer and how these layer-specific effects combine to produce productivity gains remain empirically unexplored. Motivated by this theoretical framework and empirical gap, we propose our first research question: *How and why does the AI coder affect employees' coding productivity?*

Moving beyond employees' productivity per se, we examine whether the potential AI-induced productivity gains translate into changes in work time. The theoretical prediction is not obvious. On one hand, higher productivity makes work effort more rewarding, which may incentivize employees to work longer hours to maximize their total output (Lazear 2000, Oettinger 1999, Stafford 2015). On the other hand, increased productivity allows employees to complete their tasks faster, enabling

[1]Given the workplace context of our study, we use "developers" and "employees" interchangeably for clarity.

them to stop working early and retain the saved time as leisure (Camerer et al. 1997, Crawford and Meng 2011, Thakral and Tô 2021). Recent empirical evidence from Jiang et al. (2025) demonstrates support for the first view based on national occupation data. They find that broadly, higher AI exposure is associated with longer working hours and reduced leisure time, drawing on the standard principal-agent model (Holmstrom and Milgrom 1987), which derives that workers increase effort to maximize output in response to higher marginal productivity. Our empirical setting of software development differs along critical dimensions that may vary this prediction. First, considering the output structure, software developers in workplace environments operate under scheduled output targets, characterized by product development cycles and established objectives and key results (OKRs) (Comella-Dorda et al. 2019). This institutional characteristic creates a discontinuity in the marginal benefit of effort. Second, in terms of labor market characteristics, software developers are characterized by specialized human capital, which potentially affords employees stronger bargaining power, enabling them to retain more AI-induced efficiency gains as leisure (Jiang et al. 2025). Third, current empirical tests (Jiang et al. 2025) are mainly based on AI-exposure measures, whereas we examine direct AI adoption and usage. This allows a more granular estimation of the causal effect of actual AI usage on work time, thus motivating our second research question: *How does the AI coder affect employees' work time?*

To address the research questions, we conduct empirical investigations in collaboration with a major technology company in Asia. On September 2nd, 2024, the company deployed an AI coder for all the software developers simultaneously. We leverage a regression discontinuity in time (RDiT) design around the deployment of the AI coder on randomly selected 200 software developers. This design allows us to draw causal inferences by exploiting the sharp temporal discontinuity in AI coder deployment while controlling for underlying time trends. To enhance the validity of the main findings, we also conduct robustness checks including "Donut RD", falsification tests and a series of alternative measures and specifications, which show consistent patterns.

Our analysis yields several sets of findings. First, in terms of productivity and work time, we find that AI coder increases individual coding productivity (measured by code lines) by 6.9-8.4%

while simultaneously reducing daily working hours by 3.1-4.3% without compromising code quality , which remains constant throughout the observation period. Beyond daily coding work, we also find that the AI coder significantly enhances developers' capacity in complex problem-solving (measured by ticket resolution). Second, regarding skill heterogeneity, we document a nuanced "dual effect" of AI coder across skill level groups. For productivity in daily coding work, AI acts as a "skill equalizer", benefiting lower-skilled developers more. However, for complex problem-solving capacity, AI coder acts as a "skill amplifier" that higher-skilled developers leverage to widen the performance gap.

We identify three mechanisms underlying the main productivity effects of AI coder. First, on the production side, AI coder accelerates the code production through automation on the routine tasks. Second, on the review side, AI coder reduces the time required for peer reviews, significantly lowering coordination costs. Third, on the communication side, we observe the resource reallocation: the time saved from coding and review activities is shifted toward workplace communication, evidenced by increased frequency and duration of collaborative meetings. Through a sub-sample analysis, we show that the enhanced communication is not likely driven by administrative issues but the need for collaborative problem-solving. To corroborate the mechanisms identified, we conducted an open-ended survey at the focal company, the qualitative evidence of which provides support for each mechanism.

Our paper makes several contributions to the literature on the economic value of GenAI. First, we extend the emerging literature on the productivity effects of GenAI (e.g., Noy and Zhang 2023, Brynjolfsson et al. 2025) and its application in software development (e.g., Song et al. 2024, Yeverechyahu et al. 2025) by shifting the focus from open-source environments or individual tasks to professional workplace collaboration. Conceptualizing workplace software development as a three-layer workflow, we provide evidence that productivity gains are driven not only by task automation through code production but also significantly by reduced coordination costs and enhanced team communication. Second, we advance the literature on "performance inequality" in human-AI collaboration (Wang et al. 2024, Caplin et al. 2025, Kanazawa et al. 2025) by documenting a "dual

effect" where AI acts as a "skill equalizer" for routine coding tasks while serving as a "skill amplifier" for complex problem-solving. Third, our study also contributes to the literature on technology shocks, specifically GenAI, and labor supply (Jiang et al. 2025, Aguiar et al. 2021, Green and McIntosh 2001, Galí 1999). We empirically extend the understandings of AI's impact on work time (Jiang et al. 2025) by demonstrating that AI induced productivity gains can translate into reduced working hours and it also causes a shift in work time allocation in the context of workplace software development.

2. Literature Review

2.1. AI and Productivity

The rapid emergence of GenAI, particularly large language models (LLMs), has attracted substantial research on quantifying its impact on productivity. In an early experimental study, Noy and Zhang (2023) found that ChatGPT substantially raises individual productivity in writing tasks, evidenced by the reduced time taken to finish the task and increased output quality. This pattern extends to customer service. Brynjolfsson et al. (2025) studied the staggered deployment of a GenAI-based conversational assistant to customer support agents. Their analysis revealed AI access increased productivity, measured by issues resolved per hour, by 14% on average. Similarly, Zhang and Narayandas (2025) found that AI-assisted agents responded faster and achieved greater customer sentiment improvements. The productivity gains of AI are also found in the context of taxi drivers (Kanazawa et al. 2025), and knowledge-intensive tasks, like medical coding (Wang et al. 2024), consultation (Dell'Acqua et al. 2023) and academic research (Filimonovic et al. 2025, Kusumegi et al. 2025). For instance, Filimonovic et al. (2025) demonstrated that the adoption of GenAI in academic research can bring both productivity and quality gains, measured by the higher number of published papers and journal impact factors. These studies provide evidence that GenAI can potentially serve as a powerful tool for augmentation. Some studies also document the potential adverse effects of AI on productivity, highlighting the importance of careful AI deployment considerations. For instance, Dell'Acqua et al. (2023) demonstrated that when AI capability falls short of the current task complexity, it may decrease productivity. The supervisory AI studied by

Gu et al. (2025) revealed a more complex tradeoff of AI deployment, improving service quality by 14.91% while reducing productivity by 15.80%, with differential gender effects.

The mechanisms underlying these productivity effects remain contested. Fügener et al. (2025) developed an analytical framework distinguishing between automation and augmentation roles of AI, demonstrating that optimal AI deployment depends on different types of human-AI complementarity. Their analysis reveals that the automation role increases with higher between-task complementarity, while the augmentation role increases with within-task complementarity. Empirically, Noy and Zhang (2023) found that in the writing tasks, ChatGPT mostly substitutes for worker effort rather than complementing worker skills, restructuring tasks toward idea-generation and editing and away from rough-drafting. Furthermore, Ju and Aral (2025) documented that the productivity gains may stem from the reduced coordination cost in team collaboration. Specifically, they found that compared to human-human group, human-AI teams achieved 60% greater productivity, with humans sending fewer social messages, indicating lower social coordination cost.

Beyond the direct productivity effects, recent literature also highlights its context dependence and individual heterogeneity. In drug development, Lou and Wu (2021) found that AI innovation capability affected new drug-target pair discovery, particularly for known mechanisms and medium chemical novelty, but proved less helpful for entirely novel therapeutics. Kanazawa et al. (2025) showed AI navigation assistance improved taxi driver productivity by 13.4% for low-skilled drivers by shortening cruising time, while providing minimal benefits to high-skilled drivers. In academic settings, Filimonovic et al. (2025) documented that GenAI adoption increased research productivity most pronouncedly among early-career researchers, those in technically complex subfields, and non-English-speaking authors. Wang et al. (2024) revealed that AI benefits varied systematically with experience type: workers with greater task-based experience gained more from AI, while senior workers gained less due to lower trust triggered by broader job responsibilities rather than seniority itself. Caplin et al. (2025) synthesized these patterns by showing that ability and belief calibration jointly determine AI benefits, people who accurately recognize their low ability gain the

most from AI assistance, while miscalibrated individuals fail to leverage AI effectively. Brynjolfsson et al. (2025) further consider the interaction between task routineness and human experience, showing largest productivity gains when less experienced humans working on routine problems with adequate data for AI training. In summary, the general literature demonstrates that while GenAI is a powerful productivity tool, its impact is highly contingent on the complex interplay of task context and user characteristics.

2.1.1. AI Coding Assistant Our study is closely related to the recent literature on the productivity effects of AI coders. While the growing literature has established AI's productivity-enhancing potential across diverse contexts, software development presents a unique context that warrants focused investigation. First, coding represents highly knowledge-intensive work, which requires technical expertise and cumulative problem-solving where current decisions build on prior codebases. Second, AI coder plays a role of co-creators rather than consultants, directly generating code that enters systems rather than merely offering advice, raising distinct questions about productivity attribution. Third, software development contains collaborative activities, such as mandatory peer review processes, and cross-functional coordination, which distinguish it from the more individual tasks in prior research (Noy and Zhang 2023, Brynjolfsson et al. 2025).

Research on AI coders has concentrated on open-source environments, where collaboration occurs through platforms like GitHub. In these settings, prior studies have documented productivity gains, shifts in innovation patterns and work activity compositions (e.g., Yeverechyahu et al. 2025, Song et al. 2024, Li et al. 2025, Hoffmann et al. 2025, Huang et al. 2026). For instance, regarding productivity and heterogeneity effects, Song et al. (2024) studied GitHub Copilot and observed a 5.9% increase in the project-level code contributions, though these benefits came at an 8% cost in coordination time for code integration. Their findings revealed that peripheral developers showed smaller contribution gains and faced disproportionately higher coordination costs than core developers. Li et al. (2025) introduced the concept of "cognitive arbitrage", demonstrating that AI reduces developers' cognitive and time opportunity costs in development activities, enabling

increased sustained and exploratory engagement across more varied coding activities, with particularly pronounced effects for casual developers. In terms of innovation patterns, Yeverechyahu et al. (2025) found that GitHub Copilot significantly increased the project-level contributions with notably asymmetric effects across innovation types, where iterative maintenance focused contributions increased significantly more than capability-innovation feature development. Broader literature on automation technology in open source further expands this scope by distinguishing between "mechanization" (automating routine tasks) and "orchestration" (automating coordination) in OSS workflow automation (Huang et al. 2026). They found that while mechanization accelerates routine maintenance, orchestration is essential for substantive, complex innovation. Regarding activity composition, Hoffmann et al. (2025) found that AI access induced developers to shift toward core coding activities and away from non-core project management, driven by increases in independent rather than collaborative work and exploration rather than exploitation activities. This stream of open-source literature has established several consistent patterns: AI coders substantially increase individual coding productivity, shift developer activity composition, and generate heterogeneous effects across skill and experience groups. However, these findings emerge from volunteer-driven environments characterized by flexible time allocation, self-selected project involvement, and publicly accessible codebases, offering insufficient insights on the potential outcomes of workplace adoption of AI coders.

Research examining AI coders in workplace settings remains sparse. Cui et al. (2026) stands as one of the few workplace studies examining AI coders, providing direct evidence through randomized controlled trials. They show significant productivity gains among developers using AI tools, with notably higher gains among less experienced developers. However, the underlying mechanisms remain unexplored, which might create ambiguity for organizations seeking to deploy AI coder in workplace. Our study addresses these gaps by examining AI coder deployment in a workplace setting with granular daily-level data spanning multiple performance dimensions. We theorize and empirically test three mechanisms: production-layer automation of routine coding

tasks, integration-layer reduction of coordination costs, and communication-layer resource reallocation toward collaborative problem-solving. This decomposition reveals how AI's effects propagate through the interdependent workflow system. We also present evidence on how the productivity gains may translate into the changes of employees' work time, thereby providing a holistic view of the productivity effect of AI coders in the workplace setting.

2.2. Technology and Labor Supply

Our paper is also related to the literature on technology and labor supply. Classical economic theory on the elasticity of labor supply offers us two perspectives on how AI induced productivity gains may affect employees' working time.

Neoclassical labor supply theory views labor supply as individual's labor-leisure choice, where people seek to maximize their utility from both work income and leisure (Borjas 2016, Vercherand 2014). Previous studies leveraged the theory to understand the elasticity in response to wage shocks (Lazear 2000, Fehr and Goette 2007, Oettinger 1999, Stafford 2015), reaching a consensus that when labor income increases, the marginal cost of leisure becomes higher, leading to more work time. For instance, Lazear (2000) provides the seminal evidence for this view, demonstrating that when the monetary return to effort increases, workers significantly increase output, partly through intensified effort. In contrast, a second behavioral perspective suggests that workers operate with a reference target (e.g., a specific income level or output quota). When technology or financial incentives enables them to reach this target faster, they reduce their labor supply, effectively retaining their productivity gains as leisure. This behavior has been documented in previous studies on taxi drivers (Camerer et al. 1997, Crawford and Meng 2011, Thakral and Tô 2021). Recent empirical work in the gig economy has added more nuances to this view. Allen and Lehmann-Willenbrock (2023) find that ride-hailing drivers exhibit income-targeting behavior that varies with workers' fatigue status and accumulated earnings. Chen et al. (2022) support the "income-targeting" behavior while indicate that labor supply does not increase monotonically with bonus rates. Applying these theories to AI, the predictions are ambiguous. On one hand, if AI increases the marginal productivity of labor, it

makes work efforts more rewarding (Vercherand 2014), raising the opportunity cost of leisure and incentivizing employees to work longer to maximize total output. On the other hand, if AI acts primarily as an efficiency tool that helps employees reach their output targets more quickly, they may stop working earlier.

Beyond these labor-supply theory centering around wage shocks, research also specifically examines how technological shocks alter the allocation of work time and leisure. Galí (1999) provides macro-level evidence that positive technology shocks can lead to declines in hours worked, driven by sticky prices and unchanged aggregate demand in the short run. However, there are also some studies arguing for the intensified work efforts in response to technological progress (Green and McIntosh 2001, Cantore et al. 2017). On the leisure technology side, Aguiar et al. (2021) argue that innovations in "leisure technology" (e.g., gaming and recreational computing) have drastically improved the quality of leisure time. They find that for younger cohorts, these improvements increased the reservation wage and the marginal value of leisure, contributing to a significant reduction in market working hours.

Most recently, scholars have begun to test these dynamics in the specific context of Generative AI. Jiang et al. (2025) provide evidence that AI exposure is associated with longer working hours, suggesting that employees leverage AI to expand the output of their work rather than reducing their hours, building on principal-agent framework, where the effect largely depends on whether AI complement or automate human's work and the bargaining power of the employees. However, their empirical analysis leverage occupation level AI-exposure rather than individual-level adoption, which may bring measurement error. Besides, it remains unclear whether this pattern holds in the specific knowledge-intensive context (i.e., software development), where short-term output targets (i.e., demand) are often scheduled and employees possess high bargaining power. Our study extends the literature by providing new empirical evidence on the labor-supply theory using individual-level AI adoption data in the context of software development.

3. Theoretical Framework

3.1. Conceptualizing Software Development as a Three-Layer Workflow

Software development in workplace settings is a complex, multi-stage workflow involving distinct but interdependent activities. To theorize how AI coders affect developer productivity, we first decompose the workflow of software developers into its constituent components. Recent empirical work tracking developers' daily activities reveals that software development is inherently fragmented across multiple activity categories: core coding, code review and version control processes, and collaborative communication through meetings and emails (Meyer et al. 2017). Critically, developers frequently context-switch between these activities, indicating that these activities form an interdependent workflow system rather than isolated tasks. Building on task-based theories of work organization (Autor et al. 2003, Acemoglu and Autor 2011), coordination theory (Malone and Crowston 1994) and workplace communication literature (Mroz et al. 2018), we conceptualize workplace software development as a system of three layers: production, integration, and communication. Drawing on theories of human-AI complementarity (Fügener et al. 2025), we argue that AI coders affect each layer through different collaboration modes. At the production layer, AI enables task automation by substituting for human effort on routine coding tasks. At the integration layer, AI augment human capability by reducing the cognitive and time cost required for peer code review. At the communication layer, AI indirectly augment collaborative communication through resource reallocation mechanism as time savings from the other two layers create capacity that teams reinvest in collaborative activities. Our framework enables us to theorize how AI coder works through the entire development system rather than affecting isolated tasks.

3.2. The Production Layer: Automating Routine Code Generation

The production layer encompasses individual coding activities, including the direct generation and modification of source code. Drawing on task-based frameworks (Autor et al. 2003), the production work constitutes routine and non-routine tasks based on their codifiability. Routine production tasks follow established patterns, can be decomposed into explicit procedures. Examples include

translating specifications into code, and adhering to company coding standards. Non-routine production tasks typically involve novel problem-solving and contextual judgment that rely on tacit knowledge. AI and humans demonstrate complementary strengths in the production layer. AI excels at pattern-matching and rapid generation for routine implementations, while humans excel at non-routine work requiring contextual understanding and tacit knowledge. This between-task complementarity, where AI and humans specialize in different task types, enables humans to focus on non-routine activities and employs AI to automate routine production work (Fügener et al. 2025).

The automation mechanism operates through several pathways. First, AI reduces implementation time for standard functions by providing immediate, contextually appropriate code completions that developers can accept with minimal modification. This eliminates the iterative trial-and-error process often required even for routine implementations. Second, AI reduces cognitive search costs by surfacing relevant code structures and syntax without requiring developers to recall exact syntax details or search documentation (Barke et al. 2023). This is particularly valuable for routine implementations where the developer knows what needs to be done but must look up how to express it in code.

Importantly, routine tasks automation at the production layer do not necessarily translate linearly to individual productivity because software development involves multiple stages beyond code generation. Developers may still face bottlenecks in other stages (e.g., integration and problem identification), limiting the overall productivity impact. This interdependency motivates our examination of integration and communication layers.

3.3. The Integration Layer: Reducing Coordination Cost

The integration layer encompasses the coordination processes required to merge individual contributions into the shared codebase, primarily manifested through peer code review. Code review serves as a quality assurance mechanism, where reviewers identify bugs, logic errors, and potential performance issues before code enters production (Bacchelli and Bird 2013). It also facilitates

knowledge transfer between reviewers and the code authors (Rigby and Bird 2013). More importantly, it ensures that code adheres to organizational standards and maintains consistency across systems.

From a coordination cost perspective (Malone and Crowston 1994, Srikanth and Puranam 2011), however, code review imposes significant time costs on development teams. Reviewers must context-switch from their own work to understand another developer's code changes, which requires cognitive effort proportional to code complexity and divergence from established conventions. Critically, coordination costs directly impact development cycle time. Longer review cycles might bring adverse effects on code production, as delayed integration increases the likelihood of merge conflicts and stale implementations.

Unlike the production layer where AI substitutes for human effort on routine tasks through between-task complementarity, we theorize that at the integration layer AI reduces coordination costs by improving the reviewability of code that human reviewers must process (Molison et al. 2025, Collante et al. 2025). This demonstrates a distinct form of complementarity between AI code generation and human code review: AI handles standardization and documentation work, while humans retain responsibility for evaluation and judgment. We expect that AI performs an augmentation role at the integration layer, enhancing rather than replacing human review capacity by streamlining the inputs to the review process.

AI improves code reviewability through two primary mechanisms. First, AI coders deployed on organizational codebases learn and enforce company-specific coding conventions, automatically generating code that adheres to established patterns, naming conventions, and architectural styles (Molison et al. 2025). This standardization reduces reviewers' cognitive burden and decreases the number of review iterations required before approval. Second, AI coders assist developers in generating comprehensive inline comments, function documentation, and commit messages that explain code intent (Collante et al. 2025), enabling reviewers to more quickly comprehend changes and provide focused feedback.

The cost reduction mechanism generates productivity gains by accelerating the development cycle. When code review requires less time and fewer iterations, the integration bottleneck diminishes, speeding up the overall code production. Moreover, reduced coordination costs free reviewer time that can be reallocated to other productive activities, including the communication activities we examine in the next layer.

3.4. The Communication Layer: Enabling Resource Reallocation

The communication layer encompasses synchronous meetings and discussions where multiple stakeholders engage and constitutes a substantial part of software developers' daily work. Communication activities serve several critical functions in knowledge-intensive work, especially in software development. They enable information exchange across roles in the team (Allen and Lehmann-Willenbrock 2023), for instance, combining technical insights from developers with business requirements from product managers and operational constraints from infrastructure teams. These facilitate collaborative problem-solving, where complex technical challenges are resolved through interactive discussion that merges diverse perspectives and builds shared understanding (Murugavel and Reiter-Palmon 2023). They also coordinate work dependencies, where developers negotiate resource allocation and establish implementation priorities (Mroz et al. 2018).

We theorize that AI coders augment communication indirectly through a resource reallocation mechanism. Building on task allocation framework (Fügener et al. 2025), AI can not only brings performance benefits but also creates a reallocation benefit, the freed human resources by AI can be allocated to other tasks. In workplace software development, where teams operate under fixed headcount and time constraints, the reallocation of developer time freed from the other two layers may generate substantial productivity gains.

The reallocation mechanism operates through two sequential steps. First, AI creates temporal capacity. When AI reduces time demands in the production layer (through routine task automation) and integration layer (through enhanced code reviewability), it creates temporal capacity slack that developers can reinvest in other activities. Developers can complete their production

and review work within fewer hours, releasing time previously consumed by manual implementation and review cycles. This temporal capacity represents the reallocation benefit brought by AI. Second, teams convert temporal slack into collaborative communication given potentially increased demand. Autor et al. (2003) theorize that computerization fundamentally reshapes the composition of workplace tasks. When AI handles routine implementation, teams can allocate effort toward more complex, technically ambitious projects. These complex tasks require more communication to integrate diverse expertise and align on technical approaches. AI thus creates both the capacity and the demand for enhanced communication.

Critically, AI does not substitute for communication in the way it substitutes for routine coding. Communication activities involve discussions, negotiations, and making judgment that resist AI automation (Autor 2015). AI may provide information that informs discussions (Ferreira and Li 2025), but it cannot replace the interactive, context-sensitive deliberation that characterizes effective team communication. Rather, AI indirectly complement human's capability in collaboration through enhanced communication. The reallocation mechanism has important implications for understanding AI's productivity impact. Focusing solely on substitution benefits at the production layer (faster code generation) would overlook the additional gains from reallocating freed resources toward communication-intensive problem-solving.

Together, AI coder's productivity effect emerges from the multiplicative interaction of the three layer effects. The production layer effect directly increases the rate at which code is generated through routine task automation. The integration layer effect ensures that generated code moves quickly through evaluation, preventing review bottlenecks from constraining production gains. The communication layer effect enables teams to address complex problems and dependencies that might otherwise slow or block code production. Moreover, the three-layer framework explains why workplace productivity effects may differ from those observed in open-source environments (Song et al. 2024). Open-source developers, who work with flexible time allocation and self-selected project involvement, may primarily experience production layer automation benefits. Workplace

developers, operating within organizational workflow structures, additionally benefit from review efficiency gains and communication capacity increases that emerge only through managed collaborative processes. Thus, we propose the first hypothesis as follows:

H1: *The AI coder will increase software developers' coding productivity through the combined effects of routine production automation, reduced review time, and enhanced communication.*

3.5. From Productivity Gains to Work Time Reduction

Having theorized AI's productivity effects, we further consider how these gains translate into work time outcomes. A critical question is whether AI-induced productivity improvements lead to effort expansion or time reduction. Neoclassical labor supply models (Borjas 2016, Vercherand 2014) predict that higher marginal productivity increases the opportunity cost of leisure, incentivizing workers to supply more labor hours to maximize total output. When technology makes work more productive, the wage rate per unit of effort effectively increases, making leisure relatively more expensive. Conversely, target-income models (Camerer et al. 1997, Crawford and Meng 2011) predict that workers operating with reference output targets will reduce work hours when productivity increases. When workers can reach their target output faster due to technological improvements, they stop working earlier and retain the time savings as leisure.

We argue that in workplace software development, institutional features favor time reduction. First, workplace developers operate under predetermined output targets defined by sprint goals, product roadmaps, and quarterly objectives (Comella-Dorda et al. 2019). Unlike piece-rate work where continuous output expansion is directly rewarded, workplace developers face organizational planning cycles that establish short-term output expectations. When productivity increases, developers can reach their targets faster, but expanding output substantially beyond planned targets yields limited marginal value in the short term. Workplace developers are typically salaried employees rather than piece-rate workers (lazear 2000). This compensation structure reduces the financial incentive to convert productivity gains into output expansion. Second, software developers possess specialized human capital and operate in a strong labor market characterized by high demand for

technical skills. This market position provides developers with bargaining power to capture productivity gains as reduced work hours (Jiang et al. 2025). Thus, we propose the second hypothesis as follows:

H2: *The AI coder will reduce software developers' daily working hours.*

4. Methods

4.1. Context and Data

We conducted empirical investigations in collaboration with a major technology company in Asia that specializes in software development. As a leading provider of intelligent cloud systems, the company employs software development engineers across multiple product divisions. The organization maintains a mature software development environment with standardized coding practices, version control systems, and collaborative development workflows. On September 2nd, 2024, the company deployed an LLM-powered AI coder to the software developers. This AI coder, deployed based on proprietary codebases, integrates directly into developers' integrated development environments (IDEs) and provides real-time code suggestions, auto-completion, and generation capabilities based on natural language and existing code context. The AI coder leverages large language models trained on extensive code repositories to offer contextually relevant recommendations across multiple programming languages and frameworks commonly used within the organization. Developers interact with the tool through their standard coding workflows: as they write code or input natural language descriptions of desired functionality, the AI coder generates suggestions that developers can accept, modify, or reject with simple keystrokes.

We obtained longitudinal data for 200 software developers[2] spanning four months from August 1st 2024 to November 29th 2024, encompassing both pre- and post-deployment periods of the AI coder. The dataset comprises granular records of developers' coding behaviors captured through the company's version control system, including daily metrics on code lines added, pull requests and other coding-related activities. We augmented these behavioral data with developer characteristics and their workplace trace data obtained from the company's human resources system. Our

[2]We randomly draw this sample of 200 from the company's approximately 400 employees.

main sample focuses on a time window of 30 natural days before and after the AI coder launch date (September 2nd, 2024). This focused time window enhances the validity of our regression discontinuity design by ensuring closer comparability between pre- and post-launch periods.[3] Given the workplace setting, the dataset naturally contains non-working days (i.e., weekends, holidays and cases such as leave). Note that during non-working days, developers' behavioral metrics remain undefined. This approach yields a final analysis sample of 8,040 developer-day observations.

To supplement the behavioral trace analyses and provide qualitative insight into the underlying mechanisms, we further conducted an open-ended survey with 14 software developers at the focal company after the AI coder deployment (see Appendix H). The survey covered developers' perceived impact on routine and non-routine tasks, code review experience, and changes in team communication.

4.2. Variables

Our main analysis employs multiple measures capturing developer productivity and work time. We construct these behavioral variables from the company's version control system logs, aggregating observations at the daily level for each developer. Our productivity measure is *CodeLines*, which captures the total number of lines of code added by each developer per day, including new code creation and modifications. This metric serves as a direct indicator of coding output and represents the volume of development work completed. The measure is widely used in software development research and aligns with the organizational practices for tracking developer contributions (Weber et al. 2024, Jones 1978).[4] We also consider the number of pull requests created and approved as alternative measures for productivity in robustness checks. We define work time as the total hours per day that a developer works. This variable *WorkingHours* is constructed using check-in and

[3] We also consider alternative time window and running date variable later in robustness checks.

[4] A concern in interpreting changes in code lines as coding output is the potential confounding effect of code formatting brought by AI, which could mechanically inflate line counts without reflecting productivity gains. We rule out this concern by noting that the company maintains a standardized coding format rule that remained stable throughout our entire sample period. All code committed to the version control system would be formatted according to these standards both before and after AI coder deployment, ensuring that observed changes in code lines reflect genuine coding output. Another concern is the confounder of inline comments, which may also inflate the real code output. We rule out this concern by noting that the measure of code lines only include effective code, excluding pure comments and blank lines in the code block.

check-out records from the human resource system, providing an accurate measure of developer time investment. *WorkingHours* serves as a proxy for developer work efforts and enables us to assess whether the potential productivity gains from AI coders can lead to changes of work time.

Aside from the main dependent variables, we collected additional coding related behavioral data, workplace trace data and employee demographics. We aggregated pull requests related data, including the number of submitted pull requests, the number of approved pull requests, the time taken to review a pull request, the comments received by a pull request at the employee-day level. We collected the meeting room booking records, aggregating the meeting duration into employee-day level, as the indicator for collaborative communication in workplace (Mroz et al. 2018). From the standard tickets management system, we also collected the tickets resolution data, linking resolution timestamps to specific developers. Besides, to examine the heterogeneous effects across skill groups, we obtained the skill level classification data from one month prior to our data window. The company we collaborate with employs a comprehensive evaluation metric for skill level identification, encompassing factors including but not limited to coding proficiency, technical breadth and depth, and product logic, thus representing a robust indicator for employee skill level. The descriptive statistics are presented in Table 1.

4.3. Empirical Strategy

To estimate the causal impact of AI coder on developers' behaviors, we employ a Regression Discontinuity in Time (RDiT) framework. The deployment of the AI coder occurred on September 2nd, 2024, creating a sharp temporal discontinuity in tool access for all 200 developers in our sample. We implement the RDiT design using the days to deployment date as the running variable and the deployment date threshold as a cutoff point. Moreover, to effectively apply the RDiT framework, we restrict our analysis window to a symmetric period around the implementation date, enhancing the comparability of pre- and post-deployment observations while maintaining sufficient statistical power.

Table 1 Descriptive Statistics

Variables	Description	N	Mean	SD	Min	Max
$CodeLines_{it}$	Total lines of code added on day t for developer i.	8,040	242.613	83.712	120	450
$WorkingHours_{it}$	Total working hours on day t for developer i.	8,040	7.367	0.778	5.877	9.384
PRs_{it}	Number of pull requests submitted on day t for developer i.	8,040	0.040	0.196	0	1
$ApprovedPRs_{it}$	Number of approved pull requests that are submitted on day t by developer i.	8,040	0.037	0.189	0	1
$PRReviewTime_{it}$	Average minutes taken to review pull requests that are submitted on day t by developer i.	322	10.770	3.521	4	23
$PRComments_{it}$	Number of comments received on pull requests that are submitted on day t by developer i.	322	1.481	0.525	1	3
$Meeting_{it}$	Dummy variable. One if developer i initiates any meeting on day t; zero otherwise.	8,040	0.252	0.434	0	1
$MeetingDuration_{it}$	Total minutes of the meeting initiated by developer i on day t.	8,040	18.772	36.400	0	120
$TicketsResolved_{it}$	Number of tickets resolved by developer i on day t.	8,040	0.208	0.412	0	2
$HighLevel_i$	Dummy variable. One if developer i is a higher-skilled developer; zero otherwise.	200	0.28	0.45	0	1

Notes: The sample includes 200 developers over a 61-natural-days observation window (30-natural-days window before and after the AI coder launch date on September 02, 2024). Code Lines, Working Hours, PRs, Approved PRs, Meeting, Meeting Duration, and Tickets Resolved are measured at the developer-day level. Note that during non-working days, developers' behavioral metrics remain undefined, resulting in the sample size of 8040. PR Review Time and PR Comments are measured at the pull request level, conditional on pull requests being submitted, resulting in a smaller sample size. To satisfy data privacy requirements, certain variables are transformed by multiplying original values by randomly generated constants. These transformations preserve rank ordering and percentage effects while obscuring absolute scales. All reported coefficients, descriptive statistics, and percentage changes reflect the transformed metrics.

Following established RDiT methodologies (Hausman and Rapson 2018, Cao et al. 2023, Liu and Cong 2023, Zhang et al. 2025), we incorporate polynomial time trends to account for non-linear temporal patterns. Our RDiT model specification is:

$$y_{it} = \alpha_0 + \beta AICoder_t + \sum_{k=0}^{3} \gamma_k Day_t^k + \sum_{k=0}^{3} \delta_k Day_t^k \times AICoder_t + u_i + \lambda_t + \epsilon_{it} \tag{1}$$

where y_{it} represents our outcome variables for developer i in day t, including productivity and work time. $AICoder_t$ is a binary indicator equal to one if day t occurs after the AI coder deployment date, and zero otherwise. Day_t is the running variable, calculated as the relative number of days from the AI coder deployment date. The parameter k denotes the order of the polynomial terms, which we vary from 0 to 3 to validate the robustness of our estimates. The polynomial terms $\sum_{k=0}^{3} \gamma_k Day_t^k$

flexibly capture nonlinear time trends in the outcome variables over time. The interaction terms $\sum_{k=0}^{3} \delta_k Day_t^k \times AICoder_t$ allow these trends to differ before and after AI coder deployment. We include developer fixed effects u_i to control for time-invariant individual heterogeneity, such as tenure, work style, and baseline productivity levels. We also incorporate the day-of-week fixed effects λ_t to account for weekday heterogeneity in developers' activities. ϵ_{it} is the error term, clustered at the developer level. The coefficient of interest, β, captures the average treatment effect of AI coder on developer outcomes.

The key identifying assumption underlying our RDiT design is that, absent the AI coder deployment, developer behaviors would have evolved smoothly through the deployment date without any discontinuous changes. This assumption is plausible in our context for several reasons. First, there were no other major organizational changes, policy interventions, or technology deployments during our study period that could confound our estimates. Second, the AI coder deployment was determined by organizational readiness rather than by trends in developer performance, eliminating concerns about endogenous timing. Third, the deployment occurred simultaneously for all developers in our sample, mitigating potential spillover effects or contamination.[5]

To further strengthen our identification strategy, we conduct extensive robustness checks including "Donut RD" that exclude observations immediately surrounding the deployment date to address potential "anticipation effects" or "short-term selection", as well as alternative time window, alternative running variable, falsification test and an "augmented local linear" approach[6]. Through these complementary approaches, we establish robust causal estimates of how AI coders affect software developer productivity and work time.

5. Results

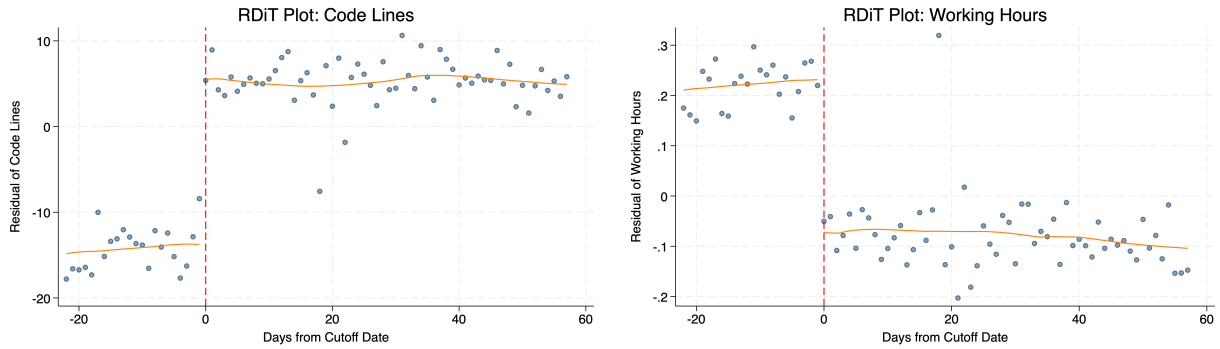
5.1. RDiT Graphs

We first show graphical evidence to validate the main empirical design using RDiT graph on the whole four-month observations (Liu and Cong 2023). The RDiT graph allows us to visually

[5]We further present the RDiT graphs later to support the assumption of smooth trend without discontinuous changes in the absence of the intervention.

[6]We present the results in robustness checks section.

assess whether there is a sharp discontinuity in developer behaviors at the deployment threshold, while accounting for confounding factors. To construct these plots, we first estimate our baseline model using Equation 1 excluding the focal treatment variable $AICoder_t$, the polynomial terms $\sum_{k=0}^{3} \gamma_k Day_t^k$ and their interactions. We then extract the residuals from this regression, which represent the component of each outcome variable that cannot be explained by other covariates. These residuals isolate the variation in outcomes that is potentially attributable to the AI coder deployment, net of other observable factors. We plot these residuals at the day level to reveal the raw pattern of outcome variables around the cutoff date to demonstrate the discontinuity. To better visualize underlying trends and reduce noise from weekday fluctuations, we add kernel regression lines with non-overlapping bins on both sides of the deployment threshold (Liu and Cong 2023). This smoothing approach provides local polynomial fits that capture the changes of outcomes in the immediate pre- and post-deployment periods.



(a) Code Lines by Day (b) Working Hours by Day

Figure 1 RDiT Graphs for Productivity and Work Time

Figure 1 presents the RDiT plots for two key outcome variables. These graphs show that after the cutoff there was a noticeable discontinuous increase in code lines, concurrent with a noticeable discontinuous decrease in working hours. Across the RDiT graphs, we observe important patterns that support our identification strategy. First, the pre-deployment trends appear smooth without anticipatory jumps immediately before week 0, suggesting that developers did not substantially

alter their behavior in anticipation of receiving the AI tool. Second, the post-deployment residuals do not exhibit unusual volatility, indicating that the effects we observe are sustained rather than reflecting short-term experimentation or novelty effects. Third, these graphical patterns provide visual evidence of discontinuities in productivity and work time, which also supports the findings of our regression discontinuity design later.

5.2. Main Effects

We focus on a narrower 30-natural-days window before and after AI coder deployment to assess its impact on developers' coding behaviors.[7] Table 2 presents the estimated effects of AI coder on developer behavioral outcomes. Column (1) shows a significantly positive coefficient for the focal variable, $AICoder$ ($\beta = 19.295, p < 0.01$), indicating that AI coder increases developers' coding productivity in terms of code lines produced. This finding remains robust across polynomial specifications in columns (2-4), with significantly positive coefficients, which represent a 6.9-8.4% productivity gain relative to the pre-treatment mean. The consistency across different polynomial orders provides strong evidence for the positive productivity effect of AI coder.

Table 2 Effects of AI Coder on Developer Productivity

	Code Lines			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	19.295***	19.362***	19.439***	16.026***
	(0.957)	(1.621)	(2.616)	(4.067)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	8,040	8,040	8,040	8,040
R-squared	0.844	0.844	0.844	0.845

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Table 3 presents the estimated effects of AI coder on developer working hours. Across all specifications in columns (1-4), we observe consistently negative and highly significant coefficients (β

[7] We also use the full time window, from Aug 1st 2024 to Nov 29th 2024 as robustness checks, the results of which are largely consistent.

ranging from -0.234 to -0.323, all $p < 0.01$), indicating that AI coder reduces working hours per day. This represents a 3.1-4.3% reduction in working hours. Overall, the findings demonstrate that developers achieve higher productivity while working for fewer hours post-deployment.

Taken together, the simultaneous increase in coding output and decrease in working hours suggests that the AI coder significantly enhances the marginal productivity of developers' work efforts. These opposite changes of both variables further indicate that in the workplace settings of software development, the productivity gains from AI are not only directed toward output expansion but are also partially retained by employees through reduced work time. This contrasts with the extended working hours documented by Jiang et al. (2025), who find that AI generally leads to longer work hours. In our setting, developers instead reduce their working time while increasing code output.

Table 3 Effects of AI Coder on Developer Work Time

	Working Hours			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	-0.287***	-0.323***	-0.234***	-0.323***
	(0.017)	(0.034)	(0.051)	(0.103)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	8,040	8,040	8,040	8,040
R-squared	0.090	0.090	0.091	0.093

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

5.3. Heterogeneous Effects

To explore whether the effects of AI coder vary across developers with different skill levels, we conduct a moderating analysis using skill level as the moderator. Table 4 presents the heterogeneous treatment effects across the key outcomes. Column (1) reveals significant heterogeneity in productivity effects across skill levels. The baseline effect for lower-level developers shows a substantial increase of 22.409 code lines per day. However, the interaction term yields a significantly negative coefficient ($\beta = -10.878, p < 0.01$), indicating that AI coder provides more pronounced productivity

gains for lower-skilled developers. Column (3) presents the effects on working hours. The interaction term is negative but statistically insignificant ($\beta = -0.012$), indicating similar AI-induced work time reduction across skill levels.

Table 4 Heterogeneous Effects of AI Coder by Skill Level

	(1)	(2)
VARIABLES	Code Lines	Working Hours
$AICoder$	22.409***	-0.320***
	(1.684)	(0.035)
$AICoder \times HighLevel$	-10.878***	-0.012
	(2.019)	(0.037)
Employee FEs	Yes	Yes
Day-of-Week FEs	Yes	Yes
BIC chosen order	1	1
Observations	8,040	8,040
R-squared	0.845	0.090

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Taken together, these findings reveal that AI coder generates asymmetric benefits across developer skill levels: lower-skilled developers experience substantially larger productivity gains, suggesting that AI coders are particularly effective at enhancing the performance of lower-skilled developers, which shows consistent patterns with prior studies (Brynjolfsson et al. 2025, Cui et al. 2026, Zhang and Narayandas 2025). However, both groups demonstrate similar reductions in working hours, indicating that the work time reduction effects of AI coders are uniformly distributed across skill levels.

5.4. Robustness Checks

In this subsection, we conduct a series of checks to validate the robustness of the main effects of AI coder on software developers' productivity.

5.4.1. Falsification Test We conduct a falsification test with alternative sample window and cutoff date to validate our main results. Specifically, we use the sample in Aug 2024, the month before actual AI coder launch and set midday of that month as pseudo cutoff date. The results are

presented in Table A1 and A2, Appendix A, which show no significant coefficients of $AICoder$ on both key outcomes, supporting our findings are not driven by spurious correlations.

5.4.2. Donut RD Although the AI coder deployment takes effect for all the users simultaneously, there are potential situations that some employees may have anticipated the deployment. In such cases, they might correspondingly change their behavior patterns around the deployment. Besides, for most users who do not anticipate the deployment, the behavior change may take time to adapt and there might be a short-term selection behavior right after the deployment. To account for the potential "anticipation effect" and "short-term selection" of the policy shock, we utilize a "Donut-RD" design (Liu and Cong 2023). Specifically, we remove the 3-day observations around the policy implementation. We present the model estimation results in Table B1 and B2, Appendix B, which are largely consistent with the main results.

5.4.3. Alternative Time Window In the main analysis, our sample covers a 30-natural-days before and after the AI coder launch. In here, we extend the sample window to the full four months time period, from Aug 1st 2024 to Nov 29th 2024, the results of which are presented in Table C1 and C2, Appendix C, demonstrating consistent results.

5.4.4. Augmented Local Linear Approach Besides, we also consider alternative "augmented local linear" approach to further utilize the long period variation pattern (Cao et al. 2023, Hausman and Rapson 2018). We first regress Equation 1 excluding the focal treatment variable $AICoder_t$, the polynomial terms $\sum_{k=0}^{3} \gamma_k Day_t^k$ and their interactions using the full sample observations (four months in total), and we record the residuals from the first step. In the second step, we regress the recorded residuals on the cutoff indicator within a shorter time window (30-natural-days before and after). We present the results of this approach in Table D1, Appendix D, demonstrating consistent patterns with our main findings.

5.4.5. Alternative Measures We re-estimate Equation 1 using alternative measures for productivity, i.e., the number of pull requests, the number of approved pull requests. The results are presented in Table E1 and E2, Appendix E, which are consistent with our main results.

5.4.6. Alternative Running Variable Besides, we also consider alternative running variables, $workday_to_cutoff$, which counts only workdays relative to cutoff date. This approach ensures the running variable increases consecutively on working days, eliminating gaps from non-working days when no metrics are recorded. The results are presented in Table F1 and F2, Appendix F. Both of them remain consistent with our main results.

6. Mechanisms

Having established the theoretical framework for how AI coders affect productivity through three distinct layers, we now empirically test these mechanisms. We examine the *Production Layer* by measuring automation intensity across skill levels, the *Integration Layer* by analyzing changes in code review processes, and the *Communication Layer* by investigating resource reallocation toward collaborative meetings.

6.1. Production Layer

We first examine whether AI coders increase productivity through automating routine coding tasks (Brynjolfsson et al. 2025, Acemoglu and Autor 2011, Goos et al. 2014), as theorized in our production layer framework. We test this hypothesis by examining whether AI disproportionately automates more work for lower-skilled developers, who typically work more on routine implementation tasks (Ide and Talamas 2025). Specifically, since routine tasks constitute a larger proportion of the workload for lower-skilled developers, we expect the automation ratio to be higher for lower-skilled group, if AI indeed primarily assists with routine tasks. To test this prediction, we decompose total code output into human-written and AI-generated components. We measure automation intensity at the developer-day level using *AI Share*, calculated as the proportion of total code that is AI-generated (AI Code Lines / Total Code Lines) for each developer-day observation. We estimate the following specification using post-deployment data only:

$$Automation_{it} = \alpha_0 + \beta_1 HighLevel_i + u_i + \lambda_t + \epsilon_{it} \tag{2}$$

where $Automation_{it}$ represents AI Share for developer i in day t (post-deployment), $HighLevel_i$ equals 1 if developer i is higher-skilled; otherwise, 0. u_i captures department and tenure fixed

effects.[8] λ_t captures the day-of-week FEs. The coefficient β_1 captures the difference in automation intensity between high-level and low-level developers.

Table 5 AI Automation by Skill Level

	(1)
VARIABLES	AI Share (%)
HighLevel	-4.159***
	(0.717)
Department FEs	Yes
Tenure FEs	Yes
Day-of-Week FEs	Yes
Observations	4,040
R-squared	0.114

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. FEs, fixed effects.

Table 5 presents the results. Column (1) shows that higher-skilled developers have significantly lower AI Share compared to lower-skilled developers ($\beta = -4.159, p < 0.01$). This indicates that AI-generated code constitutes a smaller proportion of total output for higher-skilled developers, confirming that AI coders mainly automate the routine code tasks, thus contributing to the productivity gains. This mechanism is further corroborated by our supplementary survey (Appendix H): all 14 surveyed developers reported improved efficiency specifically in routine coding tasks, with the most frequently cited benefit being the time savings in code generation and reduction of repetitive manual work. Furthermore, this automation pattern explains why AI serves as a "skill equalizer" in terms of productivity: by automating higher proportion of daily workload for lower-skilled developers, AI effectively compresses skill-based productivity differentials in code production. The technology disproportionately benefits those who previously work more with routine implementation tasks, while providing more modest gains to higher-skilled developers who typically work more on higher-level activities beyond routine code generation.

[8] In previous sections, we incorporate developer FEs directly, which however will be perfectly collinear with *HighLevel* here. Therefore, we instead incorporate department and tenure FEs.

6.2. Integration Layer

We next test whether AI coders reduce coordination costs in the code review process, as theorized in our integration layer framework. If AI-generated code improves reviewability through better adherence to standards and clearer documentation, we should observe reduced review time without sacrificing review quantity. To test this mechanism, we examine *PR Review Time*, the average time submitted pull requests take to review,[9] and *PR Review Comments*, the average number of comments received for submitted pull requests. We re-estimate Equation 1 using these measures as outcomes.

Table 6 Effects of AI Coder on Coordination Cost

	(1)	(2)
VARIABLES	PR Review Time	PR Review Comments
AICoder	-6.788**	0.464
	(2.909)	(0.305)
Employee FEs	Yes	Yes
Day-of-Week FEs	Yes	Yes
BIC chosen order	1	1
Observations	287	287
R-squared	0.442	0.287

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. Singletons were dropped in the model estimation, resulting in a smaller observation number. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Table 6 presents the results. Column (1) shows that AI coder significantly reduces PR review time ($\beta = -6.788, p < 0.05$). Column (2) shows an insignificant coefficient of AI coder on PR review comments, indicating that the reduced review time did not come at the expense of lower feedback quantity. Together, these findings validate the integration layer mechanism: AI coder accelerates the review process and reduces coordination costs, enabling faster code integration without compromising review quantity. The mechanism differs fundamentally from production-layer automation: rather than substituting for human's review efforts, AI augments human capabilities by enhancing

[9]Our measure of PR Review Time removes the waiting period between each PR's submission time and review beginning time, thus providing a more accurate representation of the coordination cost. Shorter review time indicates lower coordination cost.

code reviewability and making peer review more efficient. The open-ended survey also provides a developer-reported corroboration for this mechanism: most respondents reported being able to identify AI-generated code by characteristics that directly reduce reviewer cognitive burden, specifically, more complete comments and documentation, stricter adherence to coding standards, and more consistent code style (see Appendix H).

6.3. Communication Layer

Finally, we test whether AI coders enable resource reallocation toward collaborative communication, as theorized in our communication layer framework. Our theoretical framework posits that this reallocation operates through two sequential steps: first, AI reduces time demands in production and integration layers, creating temporal capacity slack; second, teams convert this temporal slack into collaborative communication as AI-enabled routine tasks automation allows teams to allocate effort toward more complex, technically ambitious projects that benefit from collaborative problem-solving. We expect that developers reinvest their freed capacity in meetings that facilitate collaborative problem-solving.[10]

To test this mechanism, we examine whether AI coder increases developers' engagement in workplace meetings. We measure two dimensions of communication intensity: *Meeting*, an indicator for whether developer i initiates any meeting on day t, and *Meeting Duration*, the duration of meetings developer i initiates on day t.[11] We re-estimate Equation 1 using these communication measures as outcomes.

Table 7 presents the results. Column (1) shows that AI coder significantly increases the probability of developers initiating meetings ($\beta = 0.067, p < 0.01$). Relative to the pre-deployment baseline

[10]To verify that workplace meetings in our setting primarily serve these productive technical functions, we obtained descriptive summary of the meeting records from the focal company. The company maintains internal classifications for meeting types based on meeting subjects and descriptions. Among the meetings held during our observation window, 78% were classified as technical activities spanning diverse development work, including agile iteration, planning, architecture discussions, code quality and, performance optimization. Only 22% were classified as administrative coordination (e.g., routine team meetings). We present the samples of meeting subjects in Table G1, Appendix G. This distribution confirms that meetings in software development teams facilitate the productive technical collaboration described in prior literature, rather than representing administrative overhead.

[11]We focus on meeting initiation rather than attendance for several reasons. First, initiation more directly reflects developers' active resource allocation decisions in response to AI coder deployment. Meeting attendance can be driven by passive obligations, as developers may be required to attend meetings organized by others regardless of their own capacity. Moreover, not all meeting attendances are equally meaningful: developers might attend meetings where their participation is peripheral.

Table 7 Effects of AI Coder on Meetings Initiation

	(1)	(2)
VARIABLES	Meeting	Meeting Duration
AICoder	0.067***	4.044**
	(0.024)	(2.011)
Employee FEs	Yes	Yes
Day-of-Week FEs	Yes	Yes
BIC chosen order	1	1
Observations	8,040	8,040
R-squared	0.048	0.042

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

of 0.215, this represents a 31% increase in meeting initiation frequency. Column (2) examines meeting duration and finds that AI coder increases the total time of initiated meetings by 4.044 minutes per day ($p < 0.05$), a 25.4% increase relative to the pre-deployment mean of 15.89 minutes. These results support the communication layer mechanism: AI coders create temporal capacity that developers reinvest in collaborative communication, enabling teams to address complex problems and dependencies. This finding corroborates our results in Section 7.1 showing that AI coder enhances developers' complex problem-solving capacity.

These findings reveal that unlike the direct task-level complementarity observed in production or integration, the communication layer demonstrates an indirect augmentation path where AI's effectiveness in other layers creates capacity and demand for communication. The increased meeting initiation represents developers' strategic reallocation of time saved from AI-automated routine work toward collaborative activities where human interaction remains essential (Autor 2015). This may further explain why AI enhances complex problem-solving capacity (see Section 7.1): the increased communication enables teams to tackle more technically complex problems that require integrating diverse perspectives and navigating ambiguous requirements, the contexts where human judgment and communication provide the greatest value.

One potential concern here is that the observed increase in communication may reflect heightened administrative issues rather than technical collaboration. To rule out this alternative explanation,

we examine the heterogeneity of this effect based on developers' tenure. Within our empirical setting, high-tenure employees are significantly more likely to hold managerial or administrative roles, while low-tenure employees typically serve as individual contributors focused on execution and problem-solving. If the meetings were mainly driven by non-technical administrative issues, we would expect the positive effect of AI coder on meeting initiation and duration to be pronounced among high-tenure employees who manage these processes.

Table 8 Effects of AI Coder on Meetings By Tenure

	Low Tenure		High Tenure	
	(1)	(2)	(3)	(4)
VARIABLES	Meeting	Meeting Duration	Meeting	Meeting Duration
AICoder	0.070***	4.268**	0.042	2.463
	(0.026)	(2.139)	(0.068)	(6.007)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
BIC chosen order	1	1	1	1
Observations	7,037	7,037	1,003	1,003
R-squared	0.049	0.043	0.050	0.039

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

We categorize employees based on their tenure, defining *High Tenure* employees as those whose tenure is in the top decile (90th percentile and above) of the sample distribution, and *Low Tenure* employees as the remaining 90%. We then re-estimate Equation 1 separately for these two sub-samples. Table 8 reports the results of this sub-sample analysis. We find that the aggregate increase in communication is primarily driven by the low-tenure group (Columns 1-2), who exhibit statistically significant increases in both meeting initiation frequency ($\beta = 0.070, p < 0.01$) and duration ($\beta = 4.268, p < 0.05$). Conversely, while high-tenure employees (Columns 3-4) display positive point estimates for meeting frequency ($\beta = 0.042$) and duration ($\beta = 2.463$), these coefficients are smaller in magnitude and statistically insignificant. This result argues against the alternative administrative overhead hypothesis. Instead, the fact that the effect is concentrated among low-tenure employees while high-tenure employees show only an insignificant uptake, suggests the increased

communication most probably represents technical collaboration and problem-solving, supporting the communication layer mechanism.

The open-ended survey further corroborates this resource reallocation mechanism (see Appendix H). When asked how they allocate time saved by AI, developers most commonly reported advancing to the next coding task, particularly tackling more difficult technical problems. Critically, when asked about changes in team communication, the majority described a qualitative shift in communication content, away from basic implementation details and toward requirements analysis, logic, and architectural decisions. Together, the quantitative and qualitative evidence converge on the same conclusion: the effects of AI at the production and integration layers does not only free up time but redirects developer time resource toward collaborative communication.

7. Post-hoc Analyses

7.1. Effects of AI Coder on Complex Problem-Solving Capacity

Moving beyond AI coder's effects on coding productivity, we examine its impact on developers' complex problem-solving capacity, i.e., the efficiency to resolve issues requiring sophisticated product understanding and technical troubleshooting. We operationalize this through ticket resolution: the number of tickets each developer resolves per day.

In our organizational context, tickets represent technical or service issues raised by users or operation teams for the purpose of resolving their problems while using the products (Fuchs et al. 2022, Roy et al. 2016, Gupta et al. 2018). Tickets typically span technical problems such as system and performance issues, and service problems including integration failures and configuration conflicts. For instance, a ticket might document a third-party customer's inability to integrate the focal company's authentication module with their existing system due to compatibility issues. Developer teams must diagnose root causes that extend beyond coding itself, e.g., system logic or configuration errors, and then implement appropriate solutions. Critically, tickets are managed through a shared system where developers are publicly notified and the tickets are distributed based on current work capacity, thus higher resolution counts reflect enhanced individual complex problem solving capacity.

Tickets resolution differs fundamentally from our measure of coding productivity. While productivity metrics capture code output volume in daily routine work, tickets resolution occurs irregularly and less predictably. It reflects developers' overall capacity to identify and resolve problems within complex systems, representing higher-order cognitive and technical capabilities that extend beyond routine coding tasks.

We re-estimate Equation 1, using *Tickets Resolved*, the number of tickets resolved per developer-day as the dependent variable. We present the results of AI coders' effects on developers' complex problem-solving capacity, and heterogeneous effects in Table 9. Column (1) presents the main effect of AI coder on tickets resolution. The results show a positive and significant coefficient ($\beta = 0.050$, $p < 0.05$), indicating that AI coder increases the number of incident tickets developers resolve per day. Relative to the pre-deployment mean of 0.189 tickets per developer-day, this represents a 26.5% improvement in incident resolution capacity. This finding suggests that AI coder not only enhances coding productivity but also augments developers' capacity to tackle complex problems that are technically demanding.

Table 9 Effects of AI Coder on Complex Problem-Solving Capacity

	(1)	(2)
VARIABLES	Tickets Resolved	Tickets Resolved
$AICoder$	0.050**	0.006
	(0.020)	(0.020)
$AICoder \times HighLevel$		0.155***
		(0.025)
Employee FEs	Yes	Yes
Day-of-Week FEs	Yes	Yes
BIC chosen order	1	1
Observations	8,040	8,040
R-squared	0.052	0.059

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Column (2) examines heterogeneous effects by developer skill level. In contrast to our previous findings on productivity, we observe a reversed pattern of heterogeneity in AI coder's impact

on complex problem-solving capacity. The interaction term between post-cutoff and high-level developers is positive and highly significant ($\beta = 0.155, p < 0.01$), indicating that higher-skilled developers benefit substantially more from AI coder in resolving tickets. Meanwhile, the positive but insignificant main effect suggests that lower-skilled developers experience minimal gains in tickets resolution. This reversed pattern suggests that while AI coder helps compress the gaps in overall productivity across skill groups, it amplifies skill advantages in complex problem-solving contexts. Higher-skilled developers can more effectively leverage AI coder to diagnose and implement solutions for system issues and service requests. In contrast, lower-skilled developers likely struggle to apply AI to more complex problems that require substantial domain expertise and system understanding, limiting their ability to benefit from AI assistance in this more complex domain.

7.2. Effects of AI Coder on Code Quality

Next, we examine the effects of AI coder on code quality. While the AI coder can significantly improve developers' coding productivity, it remains unclear whether these productivity gains come at the expense of code quality. We operationalize code quality through the *Approved PR Ratio*, calculated as the proportion of submitted pull requests that are finally approved and merged. This metric reflects whether code meets the company's quality standards and passes peer review. Higher approval ratios indicate that submitted code requires fewer revisions and more consistently meets quality expectations. We re-estimate Equation 1 using *Approved PR Ratio* as the dependent variable.

The estimation results are presented in Table I1, Appendix I. The results show an insignificant coefficient for AI coder on *Approved PR Ratio*, indicating that AI deployment does not significantly affect the proportion of pull requests that are approved. This suggests that the productivity increase documented earlier does not come at the cost of code quality degradation. We also examine whether quality effects vary by skill level, finding a statistically insignificant coefficient for the interaction term. This finding implies that despite the heterogeneous productivity effects where lower-skilled developers gain more, both skill groups maintain similar code quality post-deployment.

8. Discussions

8.1. Conclusions

As GenAI reshapes the landscape of knowledge work, our study provides one of the first investigations into its impact within workplace environments. Our study leverages a granular daily behavioral dataset from a major technology firm to provide causal evidence on the impact of generative AI on software developers. The empirical investigation yields three main sets of conclusions, indicating significant shifts in the nature of software development.

First, we find that AI coders increase coding productivity by 6.9-8.4% while simultaneously reduce working hours by 3.1-4.3%. This finding diverges from recent nation-level evidence suggesting that higher AI exposure leads to extended working hours (Jiang et al. 2025). Besides, we also find that AI coder significantly enhances developers' capacity in complex problem-solving. These improvements do not come at the expense of code quality, which remains constant over the whole observation period. Second, regarding the heterogeneity across skill groups, we document a "dual-effect" where AI acts as a "skill equalizer" for daily coding work but a "skill amplifier" for complex problem-solving. Specifically, lower-skilled developers accrue the largest benefits in daily coding work, whereas skilled developers leverage the tool to widen their advantage in resolving complex problems. Third, our mechanism analysis reveals that the productivity gains stem from routine work automation, reduced coordination costs and reallocated time toward workplace communication. This implies that the value of AI extends beyond individual augmentation to organizational communication enhancement. Specifically, by streamlining the code production and review process, AI reduces the routine task efforts and coordination cost inherent in collaborative knowledge work, allowing teams to shift resources to aligning on more complex, communication-intensive problems.

8.2. Theoretical Contributions

From a theoretical perspective, our paper contributes to the literature on the economic value of GenAI by investigating the deployment of an LLM-powered coding assistant within a workplace software development context.

Moreover, it advances the growing literature on AI in software development (e.g., Song et al. 2024, Yeverechyahu et al. 2025) by examining workplace rather than open-source settings. Specifically, while prior research has focused largely on volunteer-driven and flexible open source environment, we examine professional developers operating under organizational constraints and coordination requirements. We conceptualize software development in workplace as a three-layer workflow including production, integration and communication, where AI coders affect all three layers and contribute to the overall productivity. Empirically, we provide one of the first pieces of evidence that AI-induced productivity gains in the workplace are driven not only by faster code generation but significantly by reduced coordination costs and more intensive team communication, revealing a mechanism where AI streamlines the friction of collaboration rather than just individual output.

Second, we advance the literature on the "performance inequality" in human-AI collaboration (Wang et al. 2024, Caplin et al. 2025, Kanazawa et al. 2025, Brynjolfsson et al. 2025) by examining how AI coders influence developer performance across skill groups. Our work offers a comprehensive view of AI's "dual effect": acting as a "skill equalizer" for daily coding work (benefiting lower-skilled developers) while serving as a "skill amplifier" for complex problem-solving (benefiting higher-skilled developers). This finding further demonstrates that AI substitutes for experience in routine execution but complements expertise in complex problem solving.

Third, our study contributes to the literature on the technological shocks and labor supply (Aguiar et al. 2021, Jiang et al. 2025, Green and McIntosh 2001, Galí 1999), specifically by investigating how generative AI affects work time. Recent work by Jiang et al. (2025) suggests that AI exposure leads to longer working hours, based on the principal-agent framework which derives that employees continuously increase effort for output in response to higher marginal productivity. However, their empirical investigation was conducted using national occupation data with AI exposure. We extend these findings with individual-level AI adoption data in the setting of software development where short term output target (i.e., demand) is pre-determined and employees' bargaining power is high. Our result shows that the productivity gains induced by AI translate into

reduced working hours. Besides, we also document a shift in time allocation and effort composition. Specifically, we operationalize this reallocation as a shift from routine coding and review duties to collaborative communication. Our results validate that AI coders do not merely accelerate existing tasks but fundamentally alter the composition of work, allowing time saved from daily routine work to be reinvested into workplace communication.

8.3. Managerial Implications

Our paper has several implications for software development firms and broader organizations currently navigating AI deployment. First, our findings suggest that organizations should embrace AI coders not only as productivity tools but as mechanisms for employees' work-life balance. We find that AI coders increase coding output while simultaneously reducing working hours. This contrasts with the "efficiency paradox" observed in previous studies where technology extends the workday. Besides, this decoupling of code output from time input also highlights the importance of performance evaluation metrics. Organizations should transition toward outcome-based evaluation metrics (e.g., output features) rather than hours logged.

Second, our results regarding the "dual effect" of AI necessitate a nuanced approach to human resource management and task allocation. We find that AI acts as an equalizer for daily coding work (benefiting lower-skilled developers more) but as an amplifier for complex problem-solving (benefiting higher-skilled developers more). This suggests that managers should avoid a "one-size-fits-all" AI strategy. Instead, organizations should encourage a specialized division of labor: leveraging lower-skilled developers equipped with AI for routine code generation and implementation, while focusing higher-skilled developers on complex system diagnosis and architecture, where their ability to leverage AI for complex problem-solving is most distinct. This strategic alignment ensures that the firm captures the full spectrum of AI benefits.

Third, our mechanisms analysis highlights a critical shift in the nature of collaborative work. Unlike open-source contexts where AI may increase coordination overhead (Song et al. 2024), we find that in a workplace setting, AI reduces code review time and facilitates a reallocation of

resources toward collaborative communication. The observed increase in meeting frequency suggests that AI is transforming software development by shifting more significance from implementation and technical coordination to collaborative communication. Managers should explicitly support this transition by validating effective communication as a core productive activity. Rather than viewing the increased time spent in meetings as administrative waste, organizations should recognize it as a necessary reinvestment of the time saved from routine coding and reviewing, enabling teams to focus on alignment and complex decision-making and problem-solving.

8.4. Limitations and Future Work

Our paper has several limitations. First, our empirical setting is a single major technology firm in Asia. The specific organizational culture and labor norms may influence the external validity of our findings, particularly the reduction in work time. Future research should examine whether these effects persist in competitive environments with different incentive structures or in western labor markets. Second, while our observation window spans several months, sufficient to identify the immediate causal impact on productivity, we cannot rule out long-term equilibrium effects, such as potential "skill atrophy" among junior developers who may become over-reliant on AI for code generation or "learning effect", that developers become sophisticated with AI tools, exhibiting minimal skill gaps in effectively using it. Longitudinal studies spanning years are needed to assess the long-term implications. Third, we do not observe the long-term maintainability of AI-generated code. Future research should investigate the "technical debt" potential of AI coders, examining whether the speed of AI generation today creates compounded complexity for future development cycles.

Acknowledgment

The authors thank the cooperating company for providing data access, offering contextual background on the organizational practices, and facilitating the open-ended survey investigations with software developers. The authors declare no financial relationships or other conflicts of interest with any parties related to this research.

References

Acemoglu D, Autor D (2011) Skills, Tasks and Technologies: Implications for Employment and Earnings. *Handbook of Labor Economics*, volume 4, 1043–1171 (Elsevier), ISBN 978-0-444-53452-1.

Aguiar M, Bils M, Charles KK, Hurst E (2021) Leisure Luxuries and the Labor Supply of Young Men. *Journal of Political Economy* 129(2):337–382.

Allen JA, Lehmann-Willenbrock N (2023) The key features of workplace meetings: Conceptualizing the why, how, and what of meetings at work. *Organizational Psychology Review* 13(4):355–378.

Autor DH (2015) Why Are There Still So Many Jobs? The History and Future of Workplace Automation. *Journal of Economic Perspectives* 29(3):3–30.

Autor DH, Levy F, Murnane RJ (2003) The Skill Content of Recent Technological Change: An Empirical Exploration*. *The Quarterly Journal of Economics* 118(4):1279–1333.

Bacchelli A, Bird C (2013) Expectations, outcomes, and challenges of modern code review. *2013 35th International Conference on Software Engineering (ICSE)*, 712–721.

Barke S, James MB, Polikarpova N (2023) Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7(OOPSLA1):78:85–78:111.

Bick A, Blandin A, Deming D (2025) The impact of generative ai on work productivity. Technical report, Federal Reserve Bank of St. Louis.

Borjas GJ (2016) *Labor Economics*. The McGraw-Hill Series in Economics (New York, NY: McGraw-Hill Education), seventh edition.

Brynjolfsson E, Li D, Raymond L (2025) Generative AI at Work. *Quarterly Journal of Economics* qjae044.

Camerer C, Babcock L, Loewenstein G, Thaler R (1997) Labor Supply of New York City Cabdrivers: One Day at a Time. *Quarterly Journal of Economics* 112(2):407–441.

Cantore C, Ferroni F, León-Ledesma MA (2017) The dynamics of hours worked and technology. *Journal of Economic Dynamics and Control* 82:67–82.

Cao Z, Zhu Y, Li G, Qiu L (2023) Consequences of Information Feed Integration on User Engagement and Contribution: A Natural Experiment in an Online Knowledge-Sharing Community. *Information Systems Research*.

Caplin A, Deming D, Li S, Martin D, Marx P, Weidmann B, Ye KJ (2025) The ABCs of Who Benefits from Working with AI: Ability, Beliefs, and Calibration. *Management Science*.

Chen X, Li Z, Ming L, Zhu W (2022) The Incentive Game Under Target Effects in Ridesharing: A Structural Econometric Analysis. *Manufacturing & Service Operations Management* 24(2):972–992.

Chui M, Hazan E, Roberts R, Singla A, Smaje K (2023) The economic potential of generative ai.

Collante A, Abedu S, Khatoonabadi S, Abdellatif A, Alor E, Shihab E (2025) The Impact of Large Language Models (LLMs) on Code Review Process.

Comella-Dorda S, Kaur K, Zaidi A (2019) Planning in an agile organization. *McKinsey Digital* 1–10.

Crawford VP, Meng J (2011) New York City Cab Drivers' Labor Supply Revisited: Reference-Dependent Preferences with Rational-Expectations Targets for Hours and Income. *American Economic Review* 101(5):1912–1932.

Cui KZ, Demirer M, Jaffe S, Musolff L, Peng S, Salz T (2026) The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers.

Management Science.

Dell'Acqua F, McFowland E, Mollick ER, Lifshitz-Assaf H, Kellogg K, Rajendran S, Krayer L, Candelon F, Lakhani KR (2023) Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality. *SSRN Electronic Journal*.

Fehr E, Goette L (2007) Do Workers Work More if Wages Are High? Evidence from a Randomized Field Experiment. *American Economic Review* 97(1):298–317.

Ferreira D, Li J (2025) Artificial Intelligence in the Boardroom.

Filimonovic D, Rutzer C, Wunsch C (2025) Can GenAI Improve Academic Performance? Evidence from the Social and Behavioral Sciences.

Fuchs S, Drieschner C, Wittges H (2022) Improving Support Ticket Systems Using Machine Learning: A Literature Review. *Hawaii International Conference on System Sciences*.

Fügener A, Walzner DD, Gupta A (2025) Roles of Artificial Intelligence in Collaboration with Humans: Automation, Augmentation, and the Future of Work. *Management Science*.

Galí J (1999) Technology, Employment, and the Business Cycle: Do Technology Shocks Explain Aggregate Fluctuations? *American Economic Review* 89(1):249–271.

Goos M, Manning A, Salomons A (2014) Explaining Job Polarization: Routine-Biased Technological Change and Offshoring. *American Economic Review* 104(8):2509–2526.

Green F, McIntosh S (2001) The intensification of work in Europe. *Labour Economics* 8(2):291–308.

Gu W, Li M, Zhang S (2025) EXPRESS: Algorithmic Supervisor and Employee Performance. *Production and Operations Management* 10591478251331095.

Gupta M, Asadullah A, Padmanabhuni S, Serebrenik A (2018) Reducing user input requests to improve IT support ticket resolution process. *Empirical Software Engineering* 23(3):1664–1703.

Hausman C, Rapson DS (2018) Regression Discontinuity in Time: Considerations for Empirical Applications. *Annual Review of Resource Economics* 10(Volume 10, 2018):533–552.

Hoffmann M, Boysel S, Nagle F, Peng S, Xu K (2025) Generative AI and the Nature of Work.

Holmstrom B, Milgrom P (1987) Aggregation and linearity in the provision of intertemporal incentives. *Econometrica* 303–328.

Huang A, Huang N, Hong Y (2026) Workflow Automation in Open-Source Software Development: Accelerating Innovation Through Mechanization and Orchestration. *Information Systems Research*.

Ide E, Talamas E (2025) Artificial Intelligence in the Knowledge Economy. *Journal of Political Economy* 737233.

Jiang W, Park J, Xiao RJ, Zhang S (2025) AI and the Extended Workday: Productivity, Contracting Efficiency, and Distribution of Rents.

Jones TC (1978) Measuring programming quality and productivity. *IBM Systems Journal* 17(1):39–63.

Ju H, Aral S (2025) Collaborating with AI Agents: Field Experiments on Teamwork, Productivity, and Performance.

Kanazawa K, Kawaguchi D, Shigeoka H, Watanabe Y (2025) AI, Skill, and Productivity: The Case of Taxi Drivers. *Management Science*.

Kusumegi K, Yang X, Ginsparg P, de Vaan M, Stuart T, Yin Y (2025) Scientific production in the era of large language models. *Science* 390(6779):1240–1243.

Lazear EP (2000) Performance Pay and Productivity. *American Economic Review* 90(5):1346–1361.

Li X, Costello FJ, Kim K, Zhang XM (2025) Assessing Open Source Software Developer's Opportunity Costs in a Time of Generative AI.

Liu J, Cong Z (2023) The Daily Me Versus the Daily Others: How Do Recommendation Algorithms Change User Interests? Evidence from a Knowledge-Sharing Platform. *Journal of Marketing Research* 60(4):767–791.

Lou B, Wu L (2021) AI on Drugs: Can Artificial Intelligence Accelerate Drug Development? Evidence from a Large-Scale Examination of Bio-Pharma Firms. *MIS Quarterly* 45(3):1451–1482.

Luo X, Qin MS, Fang Z, Qu Z (2021) Artificial Intelligence Coaches for Sales Agents: Caveats and Solutions. *Journal of Marketing* 85(2):14–32.

Malone TW, Crowston K (1994) The interdisciplinary study of coordination. *ACM Comput. Surv.* 26(1):87–119.

Meyer AN, Barton LE, Murphy GC, Zimmermann T, Fritz T (2017) The Work Life of Developers: Activities, Switches and Perceived Productivity. *IEEE Transactions on Software Engineering* 43(12):1178–1193.

Molison AS, Moraes M, Melo G, Santos F, Assuncao WKG (2025) Is LLM-Generated Code More Maintainable & Reliable than Human-Written Code?

Mroz JE, Allen JA, Verhoeven DC, Shuffler ML (2018) Do We Really Need Another Meeting? The Science of Workplace Meetings. *Current Directions in Psychological Science* 27(6):484–491.

Murugavel VR, Reiter-Palmon R (2023) Integrating workplace meetings and team creative process literature: A multi-level perspective. *Organizational Psychology Review* 13(4):429–457.

Noy S, Zhang W (2023) Experimental evidence on the productivity effects of generative artificial intelligence. *Science*.

Oettinger GS (1999) An Empirical Analysis of the Daily Labor Supply of Stadium Vendors. *Journal of Political Economy* 107(2):360–392.

Rigby PC, Bird C (2013) Convergent contemporary software peer review practices. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 202–212 (Saint Petersburg Russia: ACM), ISBN 978-1-4503-2237-9.

Roy S, Muni DP, Tack Yan JJY, Budhiraja N, Ceiler F (2016) Clustering and Labeling IT Maintenance Tickets. Sheng QZ, Stroulia E, Tata S, Bhiri S, eds., *Service-Oriented Computing*, 829–845 (Cham: Springer International Publishing), ISBN 978-3-319-46295-0.

Song F, Agarwal A, Wen W (2024) The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot. *SSRN Electronic Journal*.

Srikanth K, Puranam P (2011) Integrating distributed work: Comparing task design, communication, and tacit coordination mechanisms. *Strategic Management Journal* 32(8):849–875.

Stafford TM (2015) What Do Fishermen Tell Us That Taxi Drivers Do Not? An Empirical Investigation of Labor Supply. *Journal of Labor Economics* 33(3):683–710.

Thakral N, Tô LT (2021) Daily Labor Supply and Adaptive Reference Points. *American Economic Review* 111(8):2417–2443.

Tong S, Jia N, Luo X, Fang Z (2021) The Janus face of artificial intelligence feedback: Deployment versus disclosure effects on employee performance. *Strategic Management Journal* 42(9):1600–

1631.

Vercherand J (2014) The Neoclassical Model of the Labour Market. Vercherand J, ed., *Labour: A Heterodox Approach*, 53–74 (London: Palgrave Macmillan UK).

Wang W, Gao GG, Agarwal R (2024) Friend or Foe? Teaming Between Artificial Intelligence and Workers with Variation in Experience. *Management Science* 70(9):5753–5775.

Weber T, Brandmaier M, Schmidt A, Mayer S (2024) Significant Productivity Gains through Programming with Large Language Models. *Proc. ACM Hum.-Comput. Interact.* 8(EICS):256:1–256:29.

Yeverechyahu D, Mayya R, Oestreicher-Singer G (2025) The Impact of Large Language Models on Open-Source Innovation: Evidence from GitHub Copilot. *SSRN Electronic Journal*.

Zhang S, Narayandas D (2025) Engaging Customers with AI in Online Chats: Evidence from a Randomized Field Experiment. *Management Science*.

Zhang X, He Q, Zhang Z (2025) Impacts of Reducing Visibility of Friends' Liked Content on User Content Engagement Across Newsfeed Channels. *Information Systems Research*.

Appendix A: Robustness Checks - Falsification Test

Table A1 Falsification Test: Effects of AI Coder on Developer Productivity

	Code Lines			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	0.827	-2.419	4.315	-6.003
	(0.914)	(1.683)	(3.794)	(8.508)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	4,400	4,400	4,400	4,400
R-squared	0.880	0.880	0.880	0.880

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Table A2 Falsification Test: Effects of AI Coder on Work Time

	Working Hours			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	0.019	-0.000	0.082	0.108
	(0.024)	(0.053)	(0.101)	(0.227)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	4,400	4,400	4,400	4,400
R-squared	0.053	0.053	0.054	0.054

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Appendix B: Robustness Checks - Donut RD

Table B1 Donut RD: Effects of AI Coder on Developer Productivity

	Code Lines			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	19.140***	19.193***	18.879***	20.456***
	(1.044)	(1.849)	(3.730)	(6.049)
Constant	232.768***	232.598***	229.116***	233.197***
	(0.511)	(1.131)	(2.448)	(4.484)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	7,440	7,440	7,440	7,440
R-squared	0.842	0.842	0.842	0.842

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Table B2 Donut RD: Effects of AI Coder on Work Time

	Working Hours			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	-0.291***	-0.348***	-0.223***	-0.450***
	(0.018)	(0.040)	(0.074)	(0.144)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	7,440	7,440	7,440	7,440
R-squared	0.092	0.092	0.093	0.095

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Appendix C: Robustness Checks - Alternative Time Window

Table C1 Full Time Window: Effects of AI Coder on Developer Productivity

	Code Lines			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	19.815***	18.346***	20.330***	16.479***
	(0.728)	(1.281)	(2.050)	(3.113)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	16,360	16,360	16,360	16,360
R-squared	0.841	0.841	0.841	0.841

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Table C2 Full Time Window: Effects of AI Coder on Work Time

	Working Hours			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	-0.306***	-0.318***	-0.294***	-0.303***
	(0.014)	(0.029)	(0.046)	(0.083)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	16,360	16,360	16,360	16,360
R-squared	0.090	0.090	0.090	0.090

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Appendix D: Robustness Checks - Augmented Local Linear Approach

Table D1 Augmented Local Linear Approach

	(1)	(2)
VARIABLES	Code Lines	Working Hours
AICoder	19.052***	-0.272***
	(0.916)	(0.016)
Observations	8,040	8,040
R-squared	0.076	0.032

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. FEs, fixed effects.

Appendix E: Robustness Checks - Alternative Measures

Table E1 Effects of AI Coder on Submitted PRs

	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	0.009***	0.060***	0.079***	0.032**
	(0.002)	(0.009)	(0.012)	(0.013)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	8,040	8,040	8,040	8,040
R-squared	0.099	0.105	0.106	0.107

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Table E2 Effects of AI Coder on Approved PRs

	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	0.013***	0.058***	0.076***	0.034***
	(0.002)	(0.008)	(0.011)	(0.012)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	8,040	8,040	8,040	8,040
R-squared	0.096	0.101	0.103	0.104

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Appendix F: Robustness Checks - Alternative Running Variable

Table F1 Alternative Running Variable: Effects of AI Coder on Developer Productivity

	Code Lines			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	19.295***	19.630***	19.426***	18.432***
	(0.957)	(1.710)	(2.397)	(3.449)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	8,040	8,040	8,040	8,040
R-squared	0.844	0.844	0.844	0.845

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Table F2 Alternative Running Variable: Effects of AI Coder on Work Time

	Working Hours			
	(1)	(2)	(3)	(4)
VARIABLES	Order 0	Order 1	Order 2	Order 3
AICoder	-0.287***	-0.328***	-0.241***	-0.331***
	(0.017)	(0.034)	(0.045)	(0.074)
Employee FEs	Yes	Yes	Yes	Yes
Day-of-Week FEs	Yes	Yes	Yes	Yes
Observations	8,040	8,040	8,040	8,040
R-squared	0.090	0.090	0.091	0.092

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.

Appendix G: Samples of Meeting Subjects

Table G1 Sample Meeting Subjects by Classification Category

Category	Sample Meeting Subjects
Panel A: Technical Meetings (78.0% of sample)	
Agile Iteration	[Product X] iteration meeting
Planning	[Product X] mid-term review and adjustment
	[Product X] retrospective and improvement discussion
Architecture	[Product X] architecture refactoring review
Code Quality	Code review key issues retrospective
	Static code analysis rule set review
Performance Optimization	[Product X] performance load testing report analysis
	[Product X] Frontend build performance optimization
Other Technical	API interface specification discussion
	Security vulnerability scan results and remediation plan
Panel B: Administration-related Meetings (22.0% of sample)	
Routine Team Meeting	Weekly team meeting
	Annual team meeting and personal development plan discussion

Notes: This table presents representative examples of meeting subjects. Meeting subjects have been anonymized to protect company confidentiality. Product names and specific identifiers have been replaced with generic placeholders. The company classified meetings based on these subjects and additional meeting descriptions maintained in their internal system.

Appendix H: Supplementary Qualitative Evidence through Open-ended Survey

We conducted an open-ended survey with 14 software developers (71.4% male, $M_{\text{age}} = 33$, age range: 28–39) from the developer team at the focal company where our behavioral trace data were collected. All participants were actively using the AI tool in their daily coding work. Survey completion took approximately 15–25 minutes on average. Below we summarize the key questions we asked developers, along with the key insights gleaned from each question.

H.1. Impact on Routine and Non-routine Coding Tasks

We first asked developers, *"How has the AI coding assistant specifically affected your efficiency in handling routine coding tasks (such as basic SQL queries, basic function implementations)? How has it influenced routine code writing time?"* All 14 developers reported improved efficiency in routine coding tasks. The most frequently mentioned impacts involved time savings in code generation and reduction of repetitive manual work. Developers emphasized that AI transformed coding from "manual search and assembly" to "natural language-driven generation", allowing them to focus more on logic and framework design rather than syntax details. Several developers (n = 8) noted that actual coding and debugging time decreased substantially, leading to overall development time reduction. Representative quotes include:

> "After introducing the AI coding assistant, efficiency in handling routine coding tasks typically shifts from 'manual search + manual assembly' to 'natural language-driven + one-click generation.' The core of efficiency improvement lies in automating low-creativity labor, allowing developers to focus on logic design." [Participant A]

> "AI coding assistants can significantly improve the writing efficiency of 'routine coding tasks,' reducing code writing time to a certain extent. Especially for variable naming and uncertain common function calls, AI coding assistants can promptly provide suggestions to enhance coding efficiency." [Participant E]

Next, we asked developers, *"For 'non-routine' or 'high-complexity' problems, how do you typically use the AI assistant? How does its helpfulness for such tasks differ from routine tasks?"* The most frequently mentioned approach involved task decomposition—manually breaking complex problems into smaller subtasks and using AI to assist while human leading the problem solving. Unlike routine tasks where AI provides "ready-to-use" solutions, for complex problems, AI serves as a "collaborative partner". A representative description of this collaborative process:

"For high-complexity problems, I decompose them into sub-problems and let AI assist with: providing thought frameworks and algorithm selection suggestions, generating code drafts for key modules, and assisting in debugging and optimizing existing code. Unlike routine tasks, AI is more of a collaborative partner in high-complexity tasks rather than a simple code generator. It can expand thinking and verify solutions, but the final architecture design and key decisions still require the developer to lead." [Participant H]

H.2. Impact on Code Review

We then asked workers whether they could perceive which code was AI-generated during code review, and if so, how such code typically performed in terms of readability and standardization. Among the 14 developers, 10 reported that they could perceive AI-generated code. The characteristics they mentioned included: more complete comments and documentation (n = 7), stricter adherence to coding standards and formatting (n = 6), more consistent code style (n = 5). Regarding code quality, the consensus was that AI-generated code generally exhibited good standardization and readability:

"AI-generated code usually strictly follows coding standards like PEP 8, with very consistent indentation, spacing, and line breaks, avoiding the format chaos common in human-written code. Basic comments are relatively rich, but variable and function names are sometimes too generic." [Participant A]

"Can perceive it. Compared to manually written code, this part has extensive comments explaining logic, and the code style is relatively consistent." [Participant B]

H.3. Resource Reallocation Mechanism

To understand the resource reallocation mechanism, we asked developers, *"Assuming the AI tool helps you complete your personal coding targets ahead of schedule, how do you typically allocate the 'saved time'? (Please list the 2–3 most common situations)"*. The most common pattern was proceeding to the next coding task, particularly complex problems (n = 10). Developers emphasized using saved time for higher-value activities, for example:

"Saved time is mainly used for: solving more challenging technical problems—deeply researching performance bottlenecks, architecture optimization, and other difficult issues; technical accumulation and sharing—writing technical documentation, organizing best practices, or conducting technical sharing with the team; code optimization and refactoring—refactoring existing code to improve maintainability and performance." [Participant H]

Finally, we asked whether developers perceived changes in team communication frequency after AI introduction, and if increased, what content the new communication focused on. Responses revealed nuanced patterns. While few developers (n = 4) reported no obvious change in communication frequency, the majority (n = 12) described qualitative shifts in communication content. The most frequently mentioned change was a shift toward technical analysis and business logic discussions:

"Communication shifted to become more about requirements discussions and logic." [Participant I]

"Discussion topics shifted from how to write code to architectural discussions." [Participant F]

"More time available to discuss complex problems and design issues." [Participant D]

Interestingly, one developer explicitly noted a reduction in low-value communication:

"Repetitive communication was significantly reduced. Some repetitive verification work has been completely handed over to AI, and team members no longer need to frequently discuss basic code problems or simple task execution details. Because low-creativity, repetitive communication decreased, communication frequency increased on high-value problem solving." [Participant A]

The open-ended survey provides strong qualitative support for the resource reallocation mechanism identified in our quantitative analysis. Developers consistently reported that AI saved time on routine coding tasks, which created temporal capacity. Developers also reported the improved standardization and readability of code generated by AI coder, which streamlines the code review procedures. Moreover, developers described a transition from low-value, repetitive communication about basic implementation details toward higher-value discussions about requirements, business logic, and system architecture. This pattern aligns with our quantitative finding that meeting time increased while coding time and code review time decreased. The qualitative data reveal the underlying mechanism: the direct impact on production and integration layer doesn't simply free up time, it enables developers to redirect time resources toward collaborative problem-solving activities that requires intensive communication.

Appendix I: Post-hoc Analyses - Effects of AI Coder on Code Quality

Table I1 Effects of AI Coder on Code Quality

	(1)	(2)
VARIABLES	Approved PR Ratio	Approved PR Ratio
$AICoder$	-0.204	-0.207
	(0.130)	(0.136)
$AICoder \times HighLevel$		0.007
		(0.071)
Employee FEs	Yes	Yes
Day-of-Week FEs	Yes	Yes
Observations	287	287
R-squared	0.285	0.285

Notes: *** $p<0.01$, ** $p<0.05$, * $p<0.1$. Robust standard errors clustered at employee level in parentheses. Singletons were dropped in the model estimation, resulting in a smaller observation number. For readability, we dropped from the table the estimation of any interaction involving the polynomial terms of Day_t, such as $Day_t \times AICoder_t$. FEs, fixed effects.