The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers*

Kevin Zheyuan Cui, Mert Demirer, Sonia Jaffe,
Leon Musolff, Sida Peng, and Tobias Salz

August 2025

Abstract

This study evaluates the effect of generative AI on software developer productivity via randomized controlled trials at Microsoft, Accenture, and an anonymous Fortune 100 company. These field experiments, run by the companies as part of their ordinary course of business, provided a random subset of developers with access to an AI-based coding assistant suggesting intelligent code completions. Though each experiment is noisy and results vary across experiments, when data is combined across three experiments and 4,867 developers, our analysis reveals a 26.08% increase (SE: 10.3%) in completed tasks among developers using the AI tool. Notably, less experienced developers had higher adoption rates and greater productivity gains.

*Cui: Princeton. Demirer: MIT. Jaffe: Microsoft. Musolff: The Wharton School of the University of Pennsylvania. Peng: Microsoft. Salz: MIT. This RCT was post-registered as AEARCTR-0014530 (Peng 2024). We are grateful to Avi Goldfarb, Shane Greenstein, Anton Korinek, Ethan Mollick, and Daniel Rock, as well as the participants of the IIOC2024 conference, ASSA2025 conference, and the AI, Cognition, and the Economy Workshop. Thanks also for data help from employees at Microsoft, GitHub, and Accenture: Phillip Coppney, Wen Du, Ya Gao, Lizzie Redford, Ryan J. Salva, Daniel A. Schocke, Amanda Silver, An-Jen Tai, Dan Tetrick, Jeff Wilcox. Mert Demirer and Tobias Salz thank the MIT GenAI initiative for funding.

1 Introduction

Many economists expect generative AI to profoundly affect the organization of economic activity (Agrawal, Gans, and Goldfarb 2019; Frank et al. 2019; Furman and Seamans 2019; Greenstein et al. 2024). Eloundou et al. (2024) estimate that generative AI can perform tasks associated with over 80% of U.S. jobs, with particularly high task coverage in occupations requiring advanced degrees. This capability—enabling AI to assist doctors in diagnosing diseases, lawyers in drafting legal documents, and software engineers in writing code—has fueled predictions of substantial productivity gains from widespread adoption (Baily, Brynjolfsson, and Korinek 2023). Others, however, are less optimistic about such productivity gains (Acemoglu 2025).

Uncertainty regarding firms' willingness to adopt these technologies and their capacity to make necessary complementary investments (Bresnahan 2024; Brynjolfsson, Rock, and Syverson 2021) makes it currently difficult to empirically assess whether or not optimism about productivity gains is justified.[1] Nevertheless, some applications of generative AI have already matured and are integrated into existing workflows. An example is software development, where coding assistants based on generative AI have gained widespread adoption.[2]

This paper investigates how generative AI affects the productivity of knowledge workers, using software developers as an example. We analyze three large-scale randomized controlled trials in real-world environments. These experiments randomly assigned access to Copilot, a coding assistant developed by GitHub in collaboration with OpenAI, to just under five thousand software developers at Microsoft, Accenture, and an anonymous Fortune 100 electronics manufacturing company (henceforth Anonymous Company). These experiments were run as part of the ordinary course of business at these companies to decide whether or how extensively to adopt these technologies, and the companies kindly shared the resulting data with us.[3] These experiments lasted two to eight months, after which all groups were granted access to Copilot.

Our preferred estimates from an instrumental variable regression suggest that usage of the coding assistant causes a 26.08% (SE: 10.3%) increase in the weekly number of completed tasks for those using the tool.[4] When we look at outcomes of secondary interest, our results support this interpretation, with a 13.55% (SE: 10.0%) increase in the number of code updates (commits) and a 38.38% (SE: 12.55%) increase in the number of times code was compiled.

A central question is whether AI offers greater benefits to low-productivity or high-

[1]In addition, it is difficult to predict further breakthroughs in the architecture of these models, which may lead to further improvements in quality or decrease the cost of inference and training.

[2]Prior academic work has shown that generative AI can pass mock interviews for coding jobs at Amazon in the top decile of human performance, can perform at human level in a database of coding challenges that measure programming logic and proficiency, and can write entire programs for simple video games from several lines of instructions (Bubeck et al. 2023). Copilot is used by 1.3 million subscribers and more than 50,000 businesses (Microsoft Corporation 2025).

[3]The implementation of these experiments was ad-hoc as they were driven by business considerations at these companies rather than research goals.

[4]Because of imperfect compliance, our preferred estimates use treatment status as an instrument for usage, so this is an estimate of the local average treatment effect for adopters.

productivity workers. Prior research finds that when workers perform similar tasks, generative AI tends to benefit lower-ability or less-experienced workers more (e.g., Brynjolfsson, Li, and Raymond 2025; Noy and Zhang 2023), though some studies suggest that the most productive workers benefit the most (Otis et al. 2024). Consistent with the former, we find that generative AI yields greater productivity gains for lower-ability workers, even when workers are performing tasks according to their tenure or seniority.

Our preferred estimates combine estimates across all three experiments and place more weight on periods with larger differences in treatment status. We make these choices because our analysis must confront challenges related to statistical power despite the large number of developers in the experiments. These challenges arise from large weekly variation in measured outcomes and factors that reduce the takeup and duration of the three experiments.[5] The experiment at Microsoft started before Copilot was widely known (and before the release of ChatGPT), and initial takeup was low. Shortly after a larger fraction of developers in the treatment group started using it, the control group was also allowed access. At Accenture, only a few hundred developers participated in the experiment. Finally, at Anonymous Company, the treatment consisted of a staggered rollout with differences in treatment status lasting only a short time.

Most studies of the impact of generative AI on worker productivity have been conducted in controlled lab-like experiments (Campero et al. 2022; Noy and Zhang 2023; Peng et al. 2023; Vaithilingam, Zhang, and Glassman 2022). In a lab-in-the-field experiment on consultants employed by Boston Consulting Group, Dell'Acqua et al. (2023) finds that productivity on 18 tasks designed to mimic the day-to-day work at a consulting company increased by 12%–25%. Evidence from these experiments generally suggests significant productivity gains from generative AI. The exception is Vaithilingam, Zhang, and Glassman (2022), who find no statistically significant difference in completion time.

While lab experiments offer a valuable opportunity to examine the short-term implications of generative AI, challenges and complex interactions arise when these tools are deployed in real-world environments (Jaffe et al. 2024). There are some observational studies of the effects of generative AI in an actual workplace setting (Hoffmann et al. 2024; Yeverechyahu, Mayya, and Oestreicher-Singer 2024). For instance, Brynjolfsson, Li, and Raymond (2025) finds that an AI-based conversational assistant increases the productivity of customer chat support agents by 14%. The drawback of these studies is the absence of random experimental assignment of these technologies.

Our work complements both the literature on lab experiments and these observational studies by studying the impact of generative AI using a field experiment in an actual workplace setting. To date, there are still few experimental studies examining the effect of generative AI in a field setting. We fill this gap in the literature by examining a field experiment with high-skilled and highly paid knowledge workers, a group that is particularly relevant given the prediction that high-skilled jobs will be most affected by this technology. Although we examine a different part of the skill distribution and use experimental variation rather than a staggered introduction, we find productivity increases similar to those reported by Brynjolfsson, Li, and Raymond (2025). Furthermore, like

[5]We observe large variation in the output of software developers due to significant heterogeneity in their seniority, with more senior managers being less likely to engage in coding activities.

them, we also find suggestive evidence that these gains are primarily driven by improved output from recent hires and employees in more junior roles. More generally, we contribute to the literature studying the productivity and on-the-job performance of software developers (Cowgill et al. 2020; Emanuel, Harrington, and Pallais 2023; Murciano-Goroff 2022).

Lastly, we contribute to an emerging literature in marketing and other fields on the broader use of large language models. These include studies that discuss whether data from large language models can be used to estimate demand (Brand, Israeli, and Ngwe 2023; Goli and Singh 2024; Gui and Toubia 2023), studies that show how to augment large language models with experimental data (Angelopoulos, Lee, and Misra 2024), and studies that demonstrate the use of large language models to select digital advertising content (Ye, Yoganarasimhan, and Zheng 2025).

2 Setting and Experiments

2.1 What Is AI-Assisted Software Development?

AI assistants for software development offer intelligent code suggestions and autocompletion within integrated development environments. As of 2024, prominent examples include GitHub Copilot, Cursor, and Replit Ghostwriter. In our study, we examine the effects of one of these tools, GitHub Copilot. GitHub Copilot was developed by GitHub in partnership with OpenAI. It was available for "technical preview" in June 2021 and publicly available in June 2022, just a few months before the first of the experiments we analyze.[6] Copilot was trained on a large corpus of code from public GitHub repositories. This allows the AI model to learn from real-world coding practices, patterns, and styles across various programming languages. See Nagle et al. (2023) for an in-depth overview of the origins and evolution of GitHub Copilot.

The landscape of coding assistants is rapidly changing even as we are writing this article. We investigate the effects of adopting the version of GitHub Copilot that existed during our experiment period, i.e., 2022-2023. This version of Copilot integrates with the software developers use for coding and acts as an intelligent autocompletion tool. As developers write code or plain text comments, Copilot analyzes the context and generates relevant code snippets, comments, and documentation. It can autocomplete code that developers might manually type or suggest snippets they would otherwise need to search for online. This capability can save developers time and potentially improve code quality by offering suggestions that the developer might not be aware of. However, like all tools based on Large Language Models (LLMs), Copilot can make mistakes. If developers rely on it without review, it could introduce errors or decrease code quality. While general-purpose LLMs like ChatGPT can also help with software development, they are less specialized and do not integrate with standard coding tools.

[6]ChatGPT, the first general-purpose public tool of this class of AI models, was released on November 30, 2022.

	Microsoft	Accenture	Anonymous Co.
Experiment Period	Sept 2022–Apr 2023	Jul 2023–Dec 2023	Sept 2023–Oct 2023
Sample Period	Jan 2022–Apr 2024	Jul 2022–Mar 2024	Jun 2023–Feb 2024
Sample Size	1,746 developers	320 developers	3,054 developers
Encouragement	Yes (Email)	Yes (Email, training, and nudges)	No
Assignment Level	Individual & team-level	Individual-level	Team-level
Design	Treatment-control randomization	Treatment-control randomization	Randomized staggered rollout

Table 1: Comparison of Key Experiment Design Features

Notes: This table contrasts the three field experiments on timing, scale, encouragement methods, assignment structure, and overall experimental design. Sample periods are longer than experiment periods as we continue to observe outcomes after everyone gains access to Copilot.

2.2 Description of Experiments

We analyze three randomized experiments conducted with software developers at Microsoft, Accenture, and Anonymous Company. In the Microsoft and Accenture experiments, one group of developers (the treated group) was randomly assigned to be able to access GitHub Copilot, whereas the other group (the control group) did not have access for eight (Microsoft) or five (Accenture) months. In the Anonymous Company experiment, all users gained access to the tool over two months, but access dates were randomized, with some teams gaining access six weeks before others. We summarize the main features of each experiment in Table 1 and discuss the details below.

Microsoft The experiment at Microsoft started in the first week of September 2022, involving 1,746 developers primarily located in the United States. Of these developers, 50.4% were randomly selected to receive access to GitHub Copilot.[7] Randomization was implemented at both the individual and the team levels.[8] In particular, 616 developers were randomized individually, and 1,130 developers were randomized at the team level, with an average team size of 6.2. The developers work on building a wide range of software within Microsoft, with tasks that include engineering, designing, and testing software products and services. They occupy various positions in the company, ranging from entry-level developers to managers. They may work in a team or individually, depending on their task and team structure.

Participants in the treated group were informed by email about the opportunity to sign up for GitHub Copilot. The email also introduced GitHub Copilot as a productivity-enhancing tool and outlined its potential impact on their coding tasks (see Figure 9 in Appendix). Beyond this email, treated participants received no specific instructions regarding their workload or workflow to ensure they used GitHub Copilot in their natural work environment. Control group participants did not receive any communication as part of the study, even when they were eventually allowed access.The experiment ended

[7] A small number of developers in the control group nevertheless were granted access to Copilot because they were working on related tools.

[8] We account for this randomization structure in calculating our standard errors below.

earlier than planned in April 2023, as growing awareness of AI-assisted coding tools led control-group participants to seek access to Copilot.

Accenture The Accenture experiment started in the last week of July 2023 and included a number of Accenture offices located in Southeast Asia. Randomization occurred at the developer level, with 61.3% of the 320 developers assigned to the treatment group. Treatment group participants were informed over email that they were eligible to sign up for GitHub Copilot. They also participated in a training session, which explained what GitHub Copilot is, how to use it, and its potential benefits.[9] Finally, the participating managers were asked to encourage the adoption of GitHub Copilot within their teams. The experiment ended in December 2023 when the control group was granted access to Copilot.

Anonymous Company The Anonymous Company experiment started in October 2023. It involved 3,054 developers who were all eventually invited to use Copilot. The invitation dates were randomized, with new invites being sent out weekly between September 2023 and October 2023.

2.3 Variables and Outcome Measures

Measuring the productivity of modern knowledge work is notoriously difficult. Our setting has the advantage that almost all professional software development follows a highly structured workflow, where specific tasks are defined and tracked through version control software. This makes internally defined goals quantifiable. All three participating organizations use the version control software GitHub. By observing the developers' GitHub activity, we can analyze many of the output metrics that are part of their workflow.

A main outcome of interest is the number of "pull requests". A pull request can be thought of as a unit of work for software developers. Within an organization, the scope of a pull request is likely to remain relatively stable over time, shaped by organizational norms and conventions, even though different organizations may define this scope differently. For instance, a pull request may ask for a feature to be added to a larger software project. A pull request will lead to a code review, often by a more senior software developer. If this review is passed, the code will be merged and thereby become part of the larger software project. We provide further details about pull requests in Appendix D.3.

We use three additional outcome variables related to the developers' workflow. Before submitting a pull request, a developer will work separately on her code, tracking smaller changes through "commits." Periodically, the developer will "build" the code they are working on, and we can observe whether it compiled successfully. Although commits and builds are not themselves final outputs, we expect them to be measures that are monotonic in the amount of work completed.

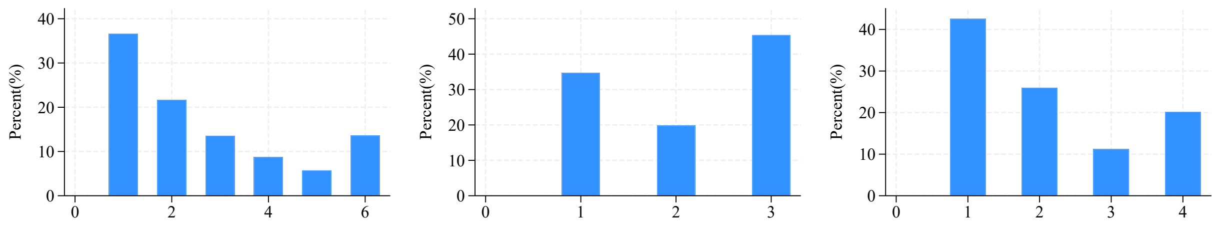
For the Microsoft experiment only, we also see some measures of code quality (such as whether a pull request was approved), which we discuss in more detail in Section 4.3

[9]This was an online training session with voluntary participation. The participants attended an online call and received instructions about what Copilot is, how to install it, how to use it, and the advantages and disadvantages of using the tool.

	Control		Treatment			
	Mean	SD	Mean	SD	Difference	p-value
Panel A: Microsoft						
Pull Requests	0.86	1.49	0.87	1.50	0.01	0.88
Commits	9.43	14.86	9.36	14.80	-0.07	0.94
Builds	7.76	12.99	7.67	12.73	-0.09	0.91
Build Success Rate	0.72	0.30	0.75	0.29	0.02	0.33
Short Tenure	0.48	0.50	0.52	0.50	0.04	0.23
Junior Level	0.55	0.50	0.61	0.49	0.06	0.03**
Panel B: Accenture						
Pull Requests	0.13	0.47	0.14	0.47	0.00	0.85
Commits	2.56	6.00	3.64	7.25	1.08	0.01**
Builds	0.96	2.54	1.10	2.68	0.14	0.38
Build Success Rate	0.51	0.37	0.54	0.38	0.03	0.40
Panel C: Anonymous						
Pull Requests	0.73	1.23	0.73	1.19	-0.00	0.99

Table 2: Balance Table

Notes: This table presents a comparison of pre-experimental outcomes in control and treatment groups across experiments. For each measure, we present its mean and standard deviation in the control group and in the treatment group. We also show the mean difference across these groups and the p-value associated with an underlying test of a difference in means. We do not present other outcome measures in Panel C because we do not have access to these data. The p-values for the differences are calculated using standard errors clustered at the level of treatment assignment, which varies across experiments (Microsoft: mixed team-level and individual assignment; Accenture: individual assignment; Anonymous Company: team-level assignment.) *10%, **5%, ***1%



(a) Microsoft (37.09% $>$ 0) (b) Accenture (12.90% $>$ 0) (c) Anonymous (34.01% $>$ 0)

Figure 1: Distribution of Pull Requests (Conditional on Above Zero)

Notes: This figure provides, for each experiment, a bar chart depicting the distribution of our primary outcome variable, the number of completed pull requests. The unit of observation is a developer-week. We plot this number after winsorizing at the 95-th percentile; its unwinsorized maximum is 892 for Microsoft, 70 for Accenture, and 876 for Anonymous. Furthermore, we condition on observations with non-zero collected pull requests.

when analyzing the effect Copilot has on them. Furthermore, again for Microsoft only, we also observe the hire date of developers and their level at the company, allowing us to separate the analysis by tenure and seniority.

In addition to these output and quality measures, we observe how developers use Copilot. For each developer who uses Copilot, we observe both the number of suggestions by Copilot and the number of accepted suggestions.

Table 2 shows summary statistics for the treatment and control groups across all three experiments, as well as balance tests for the key outcome variables. With the exception of commits in the Accenture experiment, randomization successfully balanced the average pre-treatment outcomes across the control and treatment groups. However, the table also shows that for all outcomes (with the exception of the Build Success Rate), the standard deviation exceeds the pre-treatment mean, and sometimes by a lot. This high standard deviation is driven by a large fraction of developer-weeks where the outcome variables are zero. Figure 1 shows the distribution of pull requests for weeks with at least one pull request. The high fraction of zeros, however, limits our power to detect effects in the experimental regressions below (and will be reflected in large standard errors).

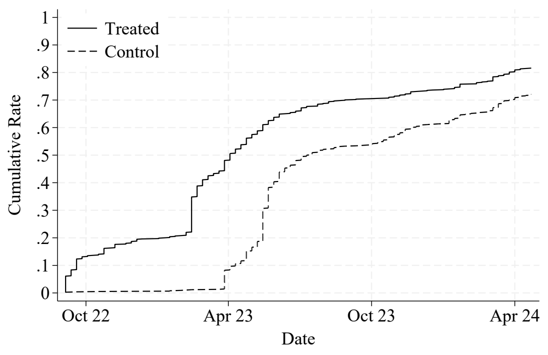
3 Adoption of Copilot

This section reports the adoption of Copilot in the experiments. Understanding adoption patterns is important for assessing the effectiveness of the experiments in generating random variation in Copilot usage. Furthermore, these patterns offer insights into the adoption of AI tools in the workplace. We define the adoption period as the first time a developer uses GitHub Copilot and consider the developer as having adopted the tool even if they later stop using it. This approach captures the initial willingness to try and use the technology.

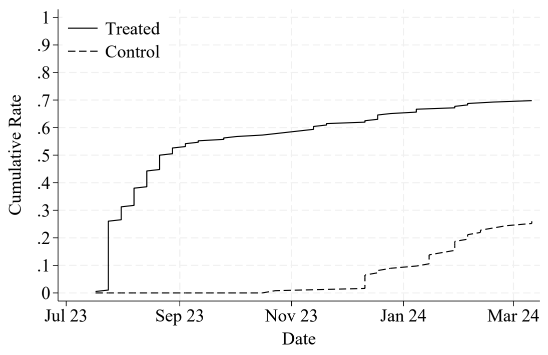
Figure 2 presents the cumulative adoption rates for the three experiments. In Panel (a), we observe that during the first two weeks of the Microsoft experiment, only 8.5 % of the treated group signed up for GitHub Copilot. This low adoption rate might have been due to inattentiveness to the initial email notification. Consequently, Microsoft sent two additional email reminders on Feb 15th, 2023, and Feb 28th, 2023. These additional emails increased the take-up rate to 42.5% within two weeks. As discussed before, the initial compliance in the control group was not perfect because a few control group developers (0.5%) required access to Copilot to work on related products. At the conclusion of the experiment in April 2023, the control group was fully given access to Copilot, and we observed rapid adoption in the control group. However, even in January 2024, adoption in the control group (64.0%) still remained below that in the treated group (75.6%), thus providing (limited) long-run variation in adoption generated by the experiment.[10]

Panel (b) reports the adoption rate of Copilot in the Accenture experiment. Unlike the Microsoft experiment, the Accenture experiment provided training in addition to encouragement emails, resulting in a faster initial uptake among treated developers. However,

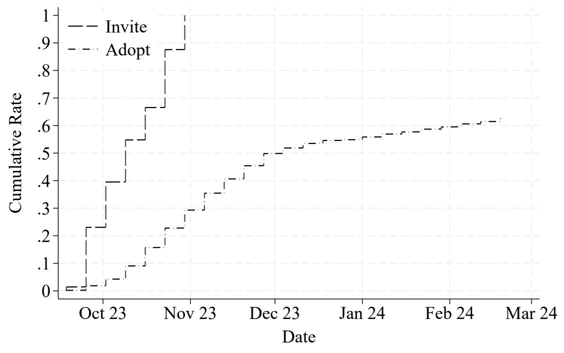
[10]This difference is also driven by the gap in Copilot access duration between the treated and control groups.



(a) Microsoft Experiment



(b) Accenture Experiment



(c) Anonymous Company Experiment

Figure 2: Cumulative Adoption Rates

Notes: The first two graphs show the cumulative rate of adoption over time for software developers in both the treatment and control groups across various experiments. In the Anonymous Company experiment (Panel c), unlike the other experiments, all developers were granted access to Copilot in a staggered fashion, with the order of access randomized among participants. Hence, we show the cumulative fraction of users invited to participate and who adopted Copilot.

within one to two months, adoption slowed down and plateaued at over 60%. Control group participants started adopting Copilot once they gained access in December 2023, but at a slower pace than the treatment group. By April 2024, the treated group's adoption rate was 69.4%, while the control group's adoption rate was 24.4%.

Panel (c) shows the staggered invitation to access Copilot (represented by the solid line) and the cumulative adoption rate among all participants (dashed line) at the Anonymous Company. As the plot shows, all developers gained access to Copilot after six weeks. During the invitation rollout in September and October, we observed a steady increase in adoption as developers gained access to Copilot. Following this initial increase in adoption, the adoption rate plateaued, exhibiting only small, steady increases for the remainder of the sample period.

It is worth noting that the adoption cost of Copilot is low relative to most AI tools in the workplace, as it integrates directly into the existing development environment and does not require any other complementary investments. Despite this, the adoption rate is significantly below 100% in all three experiments, with around 30-40% of the developers not even trying the product. Furthermore, adoption rates are notably similar across the experiments. This suggests that factors other than access, such as individual preferences and perceived quality, play important roles in developers' decisions to use this tool (Dietvorst, Simmons, and Massey 2015, 2018).[11]

4 Empirical Strategy & Main Results

4.1 Empirical Strategy

We exploit the experimental variation and address imperfect compliance by using assignment to treatment as an instrument for GitHub Copilot adoption. Hence, we estimate a local average treatment effect (LATE) that captures the impact of Copilot adoption on treatment group developers encouraged to try it.[12]

For each experiment, we observe data at the developer-week level. To gain precision, we control for both developer and week fixed effects (to account for, e.g., differences in developer skills and holidays). This leaves us with the following regression as our main specification:

$$y_{it} = \beta D_{it} + \mu_i + \gamma_t + \epsilon_{it}. \tag{1}$$

Here, β is the coefficient of interest, D_{it} is an adoption dummy that turns on after a developer first uses GitHub Copilot, μ_i is a developer fixed effect, and γ_t is a week fixed effect.[13] We estimate Equation (1) via two-stage least squares (2SLS), using an instrument Z_{it}, a dummy variable equal to one for developers randomized into treatment following the experiment's start.

Before presenting the results, we must address a key complication: the control group in our experiments was eventually granted access to GitHub Copilot. This poses no challenge for identification, but it reduces the instrument's power if we use the naive strategy detailed above. In particular, consider a hypothetical experiment that lasts just one month: at $t = 0$, developers are randomized into treatment and control groups, where control is not allowed access to Copilot until $t = 4$. Suppose further that, starting at $t = 4$, the differences in uptake between the two groups decline over time, asymptoting to zero. If we naively estimate (1) by 2SLS in this setting, the instrument's power will strictly decline with the number of observed periods. In the limit, with infinite periods, our instrument violates the relevance condition because the initial treatment assignment

[11] The significantly lower than full adoption rate observed across experiments highlights potential barriers to adoption. Although understanding the adoption of generative AI is an important question, both in itself and in the context of its productivity effects, our understanding of the reasons for adoption is limited to anecdotal evidence. Therefore, we are not able to provide a comprehensive analysis of the adoption of generative AI.

[12] If treatment effects vary and correlate with compliance, this LATE may differ from the average effect if the entire population adopted Copilot.

[13] For the initial phase of the Microsoft experiment, we do not observe intensive usage data. Hence, we say a developer at Microsoft has adopted Copilot after they either register to use it (relevant in the initial phase) or we see any usage of Copilot (relevant in the later phase).

eventually fails to predict uptake. One potential solution involves focusing only on periods where the instrument has maximal relevance ($t \in \{1, \ldots, 4\}$) to estimate the model. However, to the extent that there is still an adoption difference between treatment and control groups at $t = 5$, this strategy is wasteful in that it does not exploit all possible identifying variation.

To avoid arbitrary decisions about which post-experiment periods to include in the analysis, we weight the 2SLS estimates by the (period-by-period) difference in adoption across treatment and control groups. The resulting weighted IV regression gracefully handles the issue of declining instrument relevance, and it has been previously proposed and analyzed in the context of uptake differences across individuals (Coussens and Spiess 2021; Huntington-Klein 2020); in the context of uptake differences over time, a similar strategy was employed by Bloom et al. (2012) to improve precision. In practice, we use the Frisch-Waugh-Lovell (FWL) theorem to first clean out developer and time fixed effects from y_{it} and D_{it} in an unweighted regression, and then run a weighted IV regression of residualized $\tilde{y}_{it}$ on residualized $\tilde{D}_{it}$, instrumenting $\tilde{D}_{it}$ with Z_{it}.[14]

The impact of our weighting on the interpretation of our results is straightforward: the weighted regression weights periods based on the difference in Copilot adoption between control and treatment groups. The weighted IV estimates place greater emphasis on treatment effects during periods like March 2023 in the Microsoft experiment, where a significant difference in adoption was observed between the treatment and control groups (see Appendix Figure 11 for additional details).[15]

4.2 Results on Productivity Effects

We present our results in Table 3, split by experiment. To aid interpretation, we express coefficients as percentage effects by dividing each by the pre-treatment mean in the control group and multiplying by 100.[16] To enable easy comparison to observational studies such as Brynjolfsson, Li, and Raymond (2025), we present both difference-in-differences (DiD) estimates that do not exclusively exploit experimental variation and our main weighted IV results (W-IV), which do.[17]

At Microsoft, Copilot has a positive effect on the number of completed pull requests, commits, and code builds. Focusing on the more credible experimental estimates, however, only the effect on the number of pull requests is statistically significant at conven-

[14]Without this two-step approach, we cannot identify developer fixed effects as fixed effect identification requires within-developer variation in the instrument. However, the instrument is one in the post period for all developers, and our procedure assigns zero weights to all periods before the first adoption, hence eliminating the necessary variation for identification. Omitting developer fixed effects, in turn, would lower our power for the treatment effect. See Giles 1984 for a proof that the FWL theorem works in the context of 2SLS, though note this requires a homogeneity assumption since we use different weights in the two regressions. Alternatively, one could use only pre-treatment data to identify developer fixed effects, which requires no such assumption; if we do this, the results are quantitatively very similar.

[15]If treatment effects are homogeneous, such weighting will just improve the precision of the estimates. However, if treatment effects are heterogeneous over calendar time (e.g., because Copilot improved over time), the weighting will also affect which estimand our estimator targets.

[16]We do not take logs because a large number of person-weeks are zero for each variable. However, we report results from a Poisson difference-in-differences regression in Appendix F.

[17]We relegate the less precise unweighted results to Appendix F.

Outcome	Microsoft		Accenture		Anon. Comp.		Pooled	
	DiD	W-IV	DiD	W-IV	DiD	W-IV	DiD	W-IV
Pull Requests	7.63***	27.38**	52.65***	17.94	1.70	54.03	6.24***	26.08**
	(2.49)	(12.88)	(9.46)	(18.72)	(2.47)	(42.63)	(1.72)	(10.3)
Commits	7.03***	18.32	12.85	-4.48	-	-	7.25***	13.55
	(2.32)	(11.25)	(11.62)	(21.88)	-	-	(2.28)	(10.0)
Builds	7.11***	23.19	39.66***	92.40***	-	-	8.23***	38.38***
	(2.65)	(14.20)	(14.03)	(26.78)	-	-	(2.6)	(12.55)
Build Success	-0.65	-1.34	-20.72***	-17.40**	-	-	-1.13	-5.53
Rate	(0.79)	(4.23)	(5.06)	(7.12)	-	-	(0.78)	(3.64)

Table 3: Experiment-by-Experiment Results

Notes: This table provides difference-in-difference estimates (DiD) and weighted IV (W-IV) estimates of the effect of GitHub Copilot adoption on various productivity measures. Each entry corresponds to an estimate of β in Equation (1) expressed as a percentage of the control mean. DiD estimates instrument adoption D_{it} with itself. W-IV estimates instrument D_{it} with experimental assignment Z_{it} and weight by differences in adoption status across treatment and control (see main text). Standard errors are clustered at the level of treatment assignment, which varies across experiments (Microsoft: mixed team-level and individual assignment; Accenture: individual assignment; Anonymous Company: team-level assignment.) "Pooled" is the precision-weighted average of the other three estimates. We combine the estimates in this way instead of running a pooled regression because of the different experiment designs – the staggered rollout at Anonymous Company cannot be easily combined with the treatment/control split at the other two. *10%, **5%, ***1%

tional significance levels. We find no negative effect on the build success rate, which would decline if Copilot produced uncompilable code and developers failed to catch these errors. Although not always statistically significant, we observe directionally similar effect sizes at Accenture and Anonymous Company, except for the build success rate, which is negative at Accenture. Interestingly, the DiD estimates are sometimes larger and sometimes smaller than the experimental estimates, indicating that the reasons for the divergence between experimental and observational estimates may differ across companies.

To obtain a more precise estimate, we combine estimates across experiments in the final column, taking the precision-weighted average across the three separate estimates.[18] While standard errors are consistently large and the effect sizes differ across the three different companies, we find evidence of productivity-enhancing effects of GitHub Copilot: on average, the number of weekly pull requests made by developers increases by 26.08% (SE: 10.3%), the number of weekly commits increases by 13.55% (SE: 10.0%), and

[18] Thus, $\hat{\beta}_{Pooled} = \left(\sum_{e\in E} 1/\hat{\sigma}_e^2\right)^{-1} \sum_{e\in E} \hat{\beta}_e/\hat{\sigma}_e^2$ where $E = \{$Microsoft, Accenture, Anonymous Company$\}$, $\hat{\beta}_e$ refers to the estimate in experiment e, and $\hat{\sigma}_e$ refers to the standard error. We combine the estimates in this way instead of running a pooled regression because of the different experimental designs across the three companies – the staggered rollout at Anonymous Company cannot be easily combined with the treatment/control split at the other two.

the number of weekly builds increases by 38.38% (SE: 12.55%).

A less optimistic interpretation of the increase in builds is that developers may engage in more trial-and-error coding, accepting Copilot's suggestions and then compiling the project to check for errors. Such a change in coding style might lead to lower-quality code in the long run and undermine efficiency gains in code quantity. However, our results on build success rate only (weakly) support such an interpretation for the Accenture experiment. To investigate the quality effects further, we study various additional code quality measures we observe at Microsoft in Section 4.3 below.

We also discuss an additional experiment run at Accenture that was abandoned due to a large layoff affecting 42% of participants, resulting in a lack of data on Copilot usage and hence adoption status. Because of these data quality issues, results from this experiment are not presented in the above table and instead relegated to Appendix E. However, if we conservatively impute Copilot adoption status, we find a negative and statistically insignificant point estimate of -39.18% (SE: 36.78%) on the number of completed pull requests. The estimates on the number of commits 43.04% (SE: 38.80%) and builds 12.33% (SE: 53.60%) are both positive, though also not statistically significant. The preferred combined estimate does not change much when we include this second Accenture experiment: the number of pull requests made by developers increases by 21.34% (SE: 9.92%), the number of commits increases by 15.39% (SE: 9.69%), and the number of builds increases by 37.03% (SE: 12.22%).

4.3 Results on Code Quality

Given the evidence that Copilot access increases the number of completed tasks, it is natural to ask whether code quality may have changed as well. However, code quality is difficult to measure precisely, especially without having access to the underlying proprietary code. Instead, we investigate potential effects on code quality using several proxy outcomes available for the Microsoft experiment.

Specifically, as part of the code review process, pull requests (PRs) are evaluated by peers or supervisors. For each PR, we observe the number of reviewers and the number of reviewers who approved it. From this, we construct an approval rate—the fraction of reviewers who approved a given PR—as well as a binary indicator of whether the PR received at least one approval. We also observe the total number of comments a PR receives, which may reflect the level of controversy or the extent of revisions needed before merging. In addition, we track the number of merge conflicts, either per PR or per file, as a proxy for code modularity. Tightly coupled code that requires simultaneous changes across multiple modules is more likely to generate merge conflicts. Finally, we measure the time elapsed between a PR's initial submission and its eventual incorporation into the codebase. Individually, each of these metrics is an imperfect proxy for code quality. However, taken together, they offer suggestive evidence about how quality may have evolved.

We present our results in Table 4. We find that the approval rate of pull requests goes up by about 10%, suggesting that the code produced after adoption of Copilot is more likely to be merged into the main codebase, a positive signal of quality. All other effects are not statistically significant, but they are mostly directionally consistent: code is more

Outcome	Mean Dep	ITT	IV	Weighted IV
Approval Rate	0.60	3.78***	24.02**	9.88***
	(0.17)	(1.16)	(11.54)	(3.28)
Any Approver?	0.37	4.01**	17.83	6.87*
	(0.48)	(1.92)	(13.70)	(4.08)
# Comments Received	0.92	11.98	120.34	22.65
	(2.13)	(9.93)	(73.55)	(28.07)
# Conflicts	0.02	-3.05	-126.81	7.52
	(0.07)	(15.66)	(110.03)	(50.16)
# Conflicts Per File	0.01	1.65	-31.40	-9.50
	(0.04)	(6.80)	(44.92)	(23.92)
Hours To PR Completion	21.84	8.74	58.77	-7.02
	(48.55)	(8.05)	(55.03)	(25.27)

Table 4: Impact on Quality Measures at Microsoft.
Notes: This table investigates the effects of GitHub Copilot adoption on various measures of code quality. All quality measures are first computed PR-by-PR, and then averaged across all PRs in a developer-week; if there are no PRs in a given week, the outcome is set to missing. *Approval Rate* refers to the total number of approvers divided by the total number of reviewers for a pull request. *Any Approver?* is a dummy that is one if and only if there was at least one approver for a pull request. The # *Comments Received* refers to the total number of comments that a PR received. The # *Conflicts* refers to the number of merge conflicts, and # *Conflicts Per File* divides by the number of files touched to correct for a potential change in the scope of PRs. *Hours to PR Completion* measures how long it takes between a PR being posted and it being merged to the code base. All effects come from linear regressions, but are expressed as % of the pre-treatment mean. *10%, **5%, ***1%

likely to be approved by at least one person and creates fewer merge conflicts. However, the effect on the number of comments is positive, indicating further necessary changes to such code.

Overall, we do not find any evidence that the quality of code at Microsoft decreases after the adoption of GitHub Copilot. This contrasts with our earlier finding that, at Accenture, the Build Success Rate declined after the adoption of GitHub Copilot. Hence, we tentatively conclude that there may be heterogeneous effects on quality, though we caution that our estimates remain noisy. We believe that a more extensive study of the quality dimension is a fruitful avenue for future research.

4.4 Discussion

Before proceeding, we provide a discussion of the interpretation of our results and potential limitations.

First, due to imperfect compliance, we rely on instrumental variables (IV) estimation, which identifies a LATE. The LATE represents an average treatment effect (ATE) for a potentially selected subset of individuals—namely, those whom the encouragement designs successfully convince to adopt Copilot. The LATE may exceed the ATE if individuals with the greatest potential benefit are more responsive to encouragement (e.g., junior developers who still read all incoming emails). Conversely, it may fall below the ATE if less active developers—those who write less code and thus benefit less—are more likely to engage with such outreach. Our only empirical indication of the direction of this selection comes from our DiD estimates, which identify an average treatment effect on the treated (ATT) under the additional assumption of parallel trends. Under this assumption, the ATT is lower than the LATE at Microsoft but higher at Accenture, suggesting that the direction of selection may differ across firms.

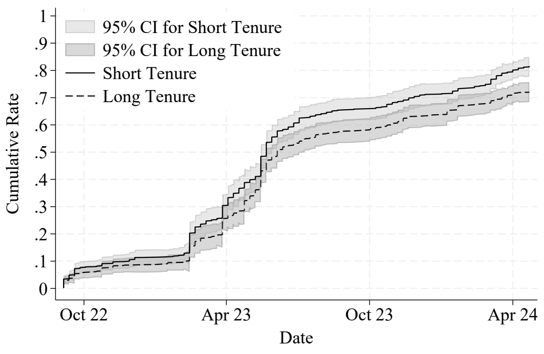
Second, while still subject to the same selection, managers may also be interested in the intent-to-treat (ITT) effect. This differs from the LATE as it reflects a weighted average of the LATE for adopters and a zero effect for non-adopters. Such ITT effects may hence reflect the effect a manager could expect from offering access to the technology without mandating adoption—though, if managers only pay for usage of these tools (usage- vs seat-based pricing), the undiluted LATE effects might be more relevant. We investigate ITT effects in Appendix B. As not all developers adopt Copilot, the ITT results are significantly lower than the LATE estimates. However, given the significant increase in adoption observed over the course of our experiments, we caution that managers may not want to use the adoption rates from early in our experiment to forecast future adoption. Indeed, given the rapid adoption of coding assistants in the industry (Bick, Blandin, and Deming 2024), the LATE estimates may serve as better benchmarks for managerially relevant productivity gains.

Third, in Appendix C we investigate whether treatment effects vary by the length of exposure to GitHub Copilot—e.g., because developers have to learn how to best use the coding assistant. While DiD estimates suggest that the effect may grow over time (consistent with learning), estimates exclusively leveraging experimental variation are underpowered to confirm this.

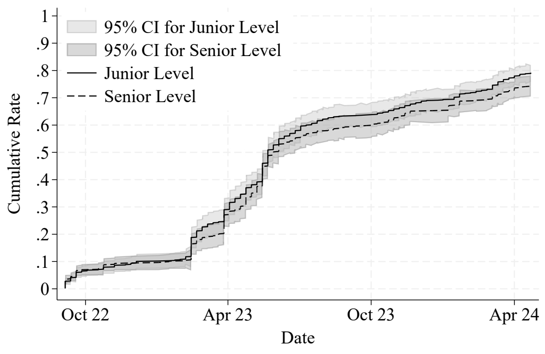
Finally, we note that estimates from different experiments vary significantly. This variation is itself interesting and could stem from various sources, such as differences in experimental design and inherent differences among companies. Although our small sample of three firms limits our ability to systematically analyze these factors, in Appendix D we provide a detailed explanation of the experiments and discuss potential sources of variation in effect sizes across companies.

5 Heterogeneity Analysis

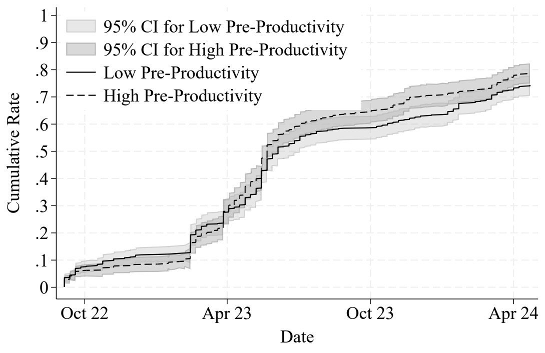
Previous literature has noted that productivity enhancements driven by large language models are heterogeneous across skill level and education. In particular, in the context of customer service and professional writing tasks, large language models have been found to help the least educated, least skilled workers the most (Brynjolfsson, Li, and Raymond 2025; Noy and Zhang 2023). In an entrepreneurial context, however, the most productive workers have been found to benefit more (Otis et al. 2024). Because we have access to developer characteristics for the Microsoft experiment, we can contribute to this open question in the literature.



(a) Adoption by Tenure



(b) Adoption by Level



(c) Adoption by Pre-Productivity

Figure 3: Heterogeneity of Adoption of Copilot

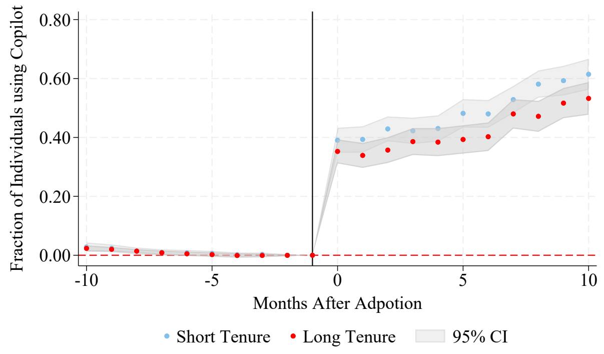
Notes: This figure explores heterogeneous adoption patterns of Copilot across developer tenure, level, and pre-period productivity. Panel (a) provides the adoption of Copilot over time, broken out by whether a developer's tenure with Microsoft at the beginning of the experiment was below or above the median; panel (b) does the same for each level. Panel (c) splits out developers with an above- vs. below-median number of pull requests before the start of the experiment.

In particular, we now break out results by (i) the tenure, (ii) the level, and (iii) the pre-experiment productivity of employees at Microsoft.[19] We split developers into short and long tenure based on the median observed tenure in our data.[20] Any developers who have been with Microsoft for less than the median time at the start of the experiment are considered "short tenure," and all other developers "long tenure." Similarly, we split developers into "junior" and "senior" based on the level at which they are employed at the company. Finally, we split developers into "low pre-productivity" and "high pre-productivity" based on the number of pre-experiment pull requests we observe from them (using the median as the cutoff).

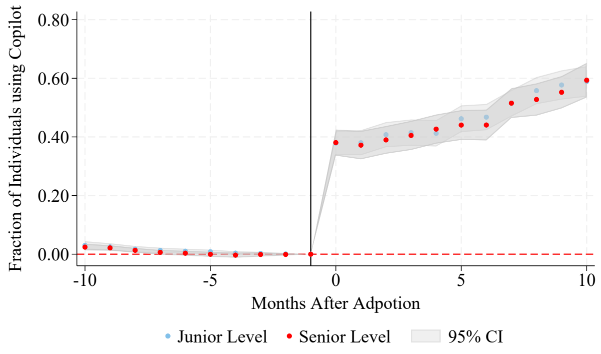
We begin our analysis by considering the heterogeneity in adoption patterns in Figure 3. We see that short-tenure developers are 9.4 percentage points (pp) (SE: 2.2pp) more

[19]We measure level as of March 1, 2023, which is the earliest date available in our data.

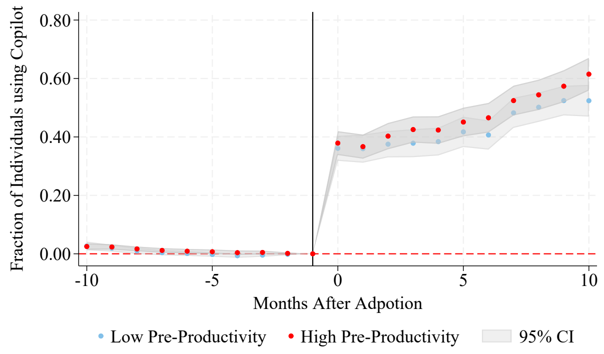
[20]As it is considered sensitive information, we cannot reveal the exact median tenure, but it is between 2 and 4 years.



(a) Usage (Since Adoption) by Tenure



(b) Usage (Since Adoption) by Level



(c) Usage (Since Adoption) by Pre-Productivity

Figure 4: Heterogeneity of Usage of Copilot

Notes: This figure explores heterogeneous usage patterns of Copilot across developer tenure, level, and pre-period productivity. We show event studies that detail the extensive margin, i.e., how likely a developer is to have used Copilot at all a given number of months after adopting Copilot; short-tenure developers are more likely to stick with Copilot.

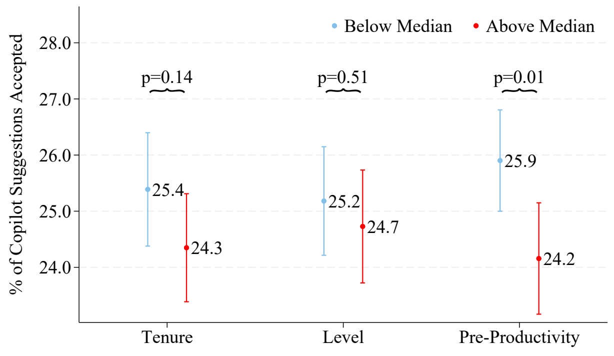
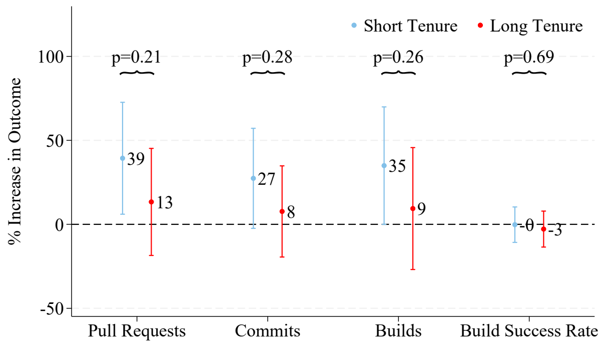
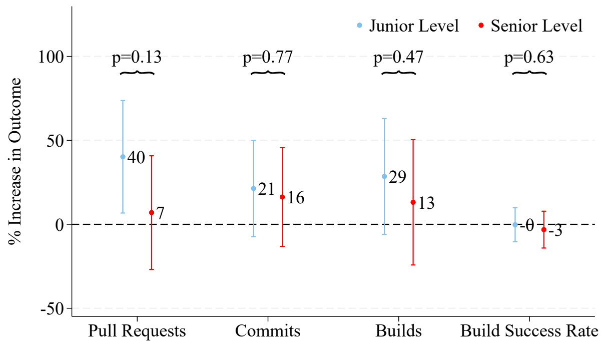


Figure 5: Heterogeneity of Fraction of Suggestions Accepted

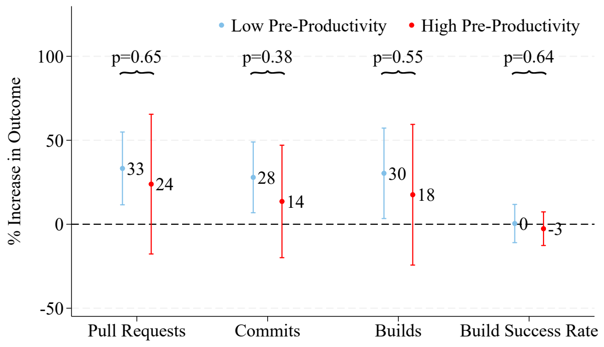
Notes: This figure shows the fraction of Copilot suggestions that developers accept; short-tenure developers are slightly and less productive developers are much more likely to accept suggestions.



(a) Treatment Effects by Tenure



(b) Treatment Effects by Level



(c) Treatment Effects by Pre-Productivity

Figure 6: Heterogeneity of Copilot Effect (Weighted IV)

Notes: This figure provides weighted IV estimates of the effect of adopting Copilot on the total number of pull requests, commits, builds, and build success rate broken out by (a) whether a developer's tenure with Microsoft at the beginning of the experiment was below median (short tenure) or above median (long tenure), (b) which level a developer was employed at and (c) the productivity of the developer before the start of the experiment. The dots in each panel are estimates derived from a single regression for each outcome where the treatment effect is allowed to differ by (a) tenure, (b) level, or (c) the developer's productivity in the pre-period as measured by his total number of completed pull requests. The bars provide 95% confidence intervals based on standard errors clustered at the level of treatment assignment. For all three outcome measures, the effects on productivity are stronger for short-tenure/junior/less productive developers, though the difference is typically not statistically significant.

likely to adopt Copilot by the end of our sample period (81.6% vs 72.1%), consistent with prior research suggesting that younger workers (who naturally have lower tenure on average) are more likely to adopt new technologies (Meyer 2011). The same effect is at play for junior developers, who are 4.7pp (SE: 2.2pp) more likely to adopt (79.2% vs. 74.4%), though the adoption difference is slightly smaller in this dimension. Intriguingly, low pre-productivity developers are initially more likely to adopt, but this changes soon after the Control group is allowed to adopt, and by the end of our sample, high pre-productivity developers are slightly more likely to adopt.

Next, Figure 4 reveals that employees of shorter tenure are more likely to continue using Copilot more than one month after initial adoption, suggesting that they perhaps expect larger benefits from the technology than their more experienced counterparts. Judging by Figure 4(b), this effect does not seem present when comparing junior to senior developers. Figure 4(c) shows that low-productivity developers are, if anything, less likely to continue using Copilot, though the difference is (just) not statistically significant (p=.085). Finally, Figure 5 reveals that higher-tenure developers are approximately 4.3% (or 1.0pp) less likely to accept Copilot's code suggestions. When comparing junior to senior developers, this difference in acceptance rates is much smaller at 1.8% (or 0.5pp), though it goes in the same direction as senior developers being less likely to accept AI suggestions. Intriguingly, there is a much larger difference across pre-productivity levels: developers who were less productive before the experiment are significantly more likely to accept any given suggestion.

Moving on to output measures, Figure 6 reports the results from weighted IV regressions.[21] The results indicate that the productivity-enhancing effects of Copilot are stronger for developers with lower tenure and those in more junior roles. While our estimates are noisy and not statistically significant at conventional levels, the pattern persists across all three main outcome measures: short-tenure developers increase their output by 27% to 39% while long-tenure developers experience smaller gains of 8% to 13%. However, we note that because of our emphasis on the average effect of *initially* adopting Copilot and the patterns in Figure 4(a), the estimates for longer-tenure developers may be attenuated by a larger number of developers abandoning the technology after an initial trial phase. Still, there is no difference in usage conditional on adoption between junior and senior developers in Figure 4(b), and we see in Figure 6(b) that junior developers increase their output by 21% to 40% while senior developers have more marginal gains of 7% to 16%. The estimates for above- and below-median productivity developers are even noisier but directionally consistent.[22]

[21]Results from unweighted IV regressions and ITT estimates can be found in the Appendix in Figures 10 and 7 respectively.

[22] We report the heterogeneous effects based on tenure and pre-experiment productivity using a quartile instead of a median split in Figure 12 in the Appendix. The results are qualitatively robust to the choice of split, though the estimates are less precise.

6 Conclusion

To summarize, we find that usage of a generative AI code suggestion tool increases software developer productivity by 26.08% (SE: 10.3%). We note that this estimate is substantially smaller than the 58% decrease Peng et al. (2023) find for the time to complete a software engineering task in the lab. It is perhaps unsurprising that the effect of AI assistance is smaller in real-world settings than in the lab, as some coding tasks may be less amenable to Copilot's assistance. Moreover, since coding is only one part of a developer's responsibilities, the time saved on coding may not fully translate into additional coding output. Our estimate is based on observing, partly over multiple years, the output of almost five thousand software developers at three different companies as part of their regular job, which strongly supports its external validity.

References

Acemoglu, Daron (2025). "The Simple Macroeconomics of AI". In: *Economic Policy* 40.121, pp. 13–58.

Agrawal, Ajay, Joshua Gans, and Avi Goldfarb (2019). "Economic Policy for Artificial Intelligence". In: *Innovation Policy and the Economy* 19.1, pp. 139–159.

Angelopoulos, Panagiotis, Kevin Lee, and Sanjog Misra (2024). "Causal Alignment: Augmenting Language Models With A/B Tests". In: *SSRN Working Paper, 4781850*.

Baily, Martin Neil, Erik Brynjolfsson, and Anton Korinek (2023). "Machines of Mind: The Case for an AI-Powered Productivity Boom". In: *Brookings Institute*.

Bick, Alexander, Adam Blandin, and David J Deming (2024). "The Rapid Adoption of Generative AI". In: *NBER Working Paper, w32966*.

Bloom, Nicholas, Benn Eifert, Aprajit Mahajan, David McKenzie, and John Roberts (2012). "Does Management Matter? Evidence From India". In: *The Quarterly Journal of Economics* 128.1, pp. 1–51.

Brand, James, Ayelet Israeli, and Donald Ngwe (2023). "Using GPT for Market Research". In: *Harvard Business School Marketing Unit Working Paper* 23-062.

Bresnahan, Timothy (2024). "What Innovation Paths for AI to Become a GPT?" In: *Journal of Economics & Management Strategy* 33.2, pp. 305–316.

Brynjolfsson, Erik, Danielle Li, and Lindsey Raymond (2025). "Generative AI at Work". In: *The Quarterly Journal of Economics* 140.2, pp. 889–942.

Brynjolfsson, Erik, Daniel Rock, and Chad Syverson (2021). "The Productivity J-Curve: How Intangibles Complement General Purpose Technologies". In: *American Economic Journal: Macroeconomics* 13.1, pp. 333–372.

Bubeck, Sébastien, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. (2023). "Sparks of Artificial General Intelligence: Early Experiments With GPT-4". In: *arXiv Preprint arXiv:2303.12712*.

Campero, Andres, Michelle Vaccaro, Jaeyoon Song, Haoran Wen, Abdullah Almaatouq, and Thomas W. Malone (2022). *A Test for Evaluating Performance in Human-Computer Systems*. arXiv: 2206.12390.

Coussens, Stephen and Jann Spiess (2021). *Improving Inference From Simple Instruments Through Compliance Estimation*. arXiv: 2108.03726.

Cowgill, Bo, Fabrizio Dell'Acqua, Samuel Deng, Daniel Hsu, Nakul Verma, and Augustin Chaintreau (2020). "Biased Programmers? Or Biased Data? A Field Experiment in Operationalizing AI Ethics". In: *Proceedings of the 21st ACM Conference on Economics and Computation*, pp. 679–681.

Dell'Acqua, Fabrizio, Edward McFowland, Ethan R. Mollick, Hila Lifshitz-Assaf, Katherine Kellogg, Saran Rajendran, Lisa Krayer, François Candelon, and Karim R. Lakhani (2023). "Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality". In: *Harvard Business School Technology & Operations Mgt. Unit Working Paper* 24-013.

Dietvorst, Berkeley J, Joseph P Simmons, and Cade Massey (2015). "Algorithm Aversion: People Erroneously Avoid Algorithms After Seeing Them Err." In: *Journal of Experimental Psychology: General* 144.1, p. 114.

— (2018). "Overcoming Algorithm Aversion: People Will Use Imperfect Algorithms if They Can (Even Slightly) Modify Them". In: *Management Science* 64.3, pp. 1155–1170.

Eloundou, Tyna, Sam Manning, Pamela Mishkin, and Daniel Rock (2024). "GPTs Are GPTs: Labor Market Impact Potential of LLMs". In: *Science* 384.6702, pp. 1306–1308.

Emanuel, Natalia, Emma Harrington, and Amanda Pallais (2023). "The Power of Proximity to Coworkers: Training for Tomorrow or Productivity Today?" In: *NBER Working Paper, w31880*.

Frank, Morgan R, David Autor, James E Bessen, Erik Brynjolfsson, Manuel Cebrian, David J Deming, Maryann Feldman, Matthew Groh, José Lobo, Esteban Moro, et al. (2019). "Toward Understanding the Impact of Artificial Intelligence on Labor". In: *Proceedings of the National Academy of Sciences* 116.14, pp. 6531–6539.

Furman, Jason and Robert Seamans (2019). "AI and the Economy". In: *Innovation Policy and the Economy* 19.1, pp. 161–191.

Giles, David E.A. (1984). "Instrumental Variables Regressions Involving Seasonal Data". In: *Economics Letters* 14.4, pp. 339–343.

Goli, Ali and Amandeep Singh (2024). "Frontiers: Can Large Language Models Capture Human Preferences?" In: *Marketing Science* 43.4, pp. 709–722.

Greenstein, Shane, Nathaniel Lovin, Scott Wallsten, Kerry Herman, and Susan Pinckney (2024). "A Guide to the Vocabulary, Evolution, and Impact of Artificial Intelligence (AI)". In: *Harvard Business School Working Paper* 625-039.

Gui, George and Olivier Toubia (2023). "The Challenge of Using LLMs to Simulate Human Behavior: A Causal Inference Perspective". In: *arXiv Preprint arXiv:2312.15524*.

Hoffmann, Manuel, Sam Boysel, Frank Nagle, Sida Peng, and Kevin Xu (2024). "Generative AI and the Nature of Work". In: *CESifo Working Paper*.

Huntington-Klein, Nick (2020). "Instruments With Heterogeneous Effects: Bias, Monotonicity, and Localness". In: *Journal of Causal Inference* 8.1, pp. 182–208.

Jaffe, Sonia, Neha Parikh Shah, Jenna Butler, Alex Farach, Alexia Cambon, Brent Hecht, Michael Schwarz, and Jaime Teevan (2024). "Generative AI in Real-World Workplaces". In: *Microsoft*.

Meyer, Jeremy (2011). "Workforce Age and Technology Adoption in Small and Medium-Sized Service Firms". In: *Small Business Economics* 37.3, pp. 305–324.

Microsoft Corporation (2025). *Microsoft Fiscal Year 2024 Q2 Earnings Call*. `https://www.microsoft.com/en-us/investor/events/fy-2024/earnings-fy-2024-q2`. Conference Call Transcript and Webcast.

Murciano-Goroff, Raviv (2022). "Missing Women in Tech: The Labor Market for Highly Skilled Software Engineers". In: *Management Science* 68.5, pp. 3262–3281.

Nagle, F, S Greenstein, M Roche, NL Wright, and S Mehta (2023). "CoPilots(s): Generative AI at Microsoft and GitHub". In: *Harvard Business School Case* 9, pp. 624–010.

Noy, Shakked and Whitney Zhang (2023). "Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence". In: *Science* 381.6654, pp. 187–192.

Otis, Nicholas, Rowan Clarke, Solène Delecourt, David Holtz, and Rembrand Koning (2024). "The Uneven Impact of Generative AI on Entrepreneurial Performance". In: *Harvard Business School Working Paper, 24-042*.

Peng, Sida (2024). *The Effects of Generative AI on High Skilled Work: Evidence From Three Field Experiments With Software Developers*. AEA RCT Registry.

Peng, Sida, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer (2023). "The Impact of AI on Developer Productivity: Evidence From GitHub Copilot". In: *arXiv Preprint arXiv:2302.06590*.

Vaithilingam, Priyan, Tianyi Zhang, and Elena L Glassman (2022). "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models". In: *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pp. 1–7.

Ye, Zikun, Hema Yoganarasimhan, and Yufeng Zheng (2025). "LOLA: LLM-Assisted Online Learning Algorithm for Content Experiments". In: *Marketing Science* 0.0.

Yeverechyahu, Doron, Raveesh Mayya, and Gal Oestreicher-Singer (2024). "The Impact of Large Language Models on Open-Source Innovation: Evidence From GitHub Copilot". In: *arXiv Preprint arXiv:2409.08379*.

The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers

Kevin Zheyuan Cui Mert Demirer
Sonia Jaffe Leon Musolff Sida Peng Tobias Salz

Appendix - For Online Publication

A Data Cleaning

We provide details on which individuals we had to exclude from each raw dataset and the reasons for their exclusion.

A.1 Microsoft

In the original sample of the dataset, we have 1,746 individuals. We kept only software developers, which leaves us with 1,538, and we dropped people who switched organizations, leaving us 1,522. Finally, we dropped one individual who adopted Copilot before the experiment started, with a final sample of 1,521.

We also drop the data for the last week of the dataset since the dataset does not record the full week of activity for the last week.

Finally, note that while the restriction for the control group was lifted in April 2023, ten individuals in the control group adopted before that date. We include these individuals in our regressions, which naturally weakens the instrument's strength.

A.2 Accenture

We drop individuals who have no record of data and people who have left the company. We start with the original dataset containing 369 individuals. After dropping individuals with no outcome measures, we are left with 320. After further dropping individuals who left the company, we are left with a final sample of 316.

Finally, we note that while individuals in the control group were allowed to adopt starting December 2023, there was one individual in the control group who adopted in October 2023. We include this individual in our regressions.

A.3 Anonymous Company

The original sample has 3,054 individuals. We drop individuals who have shown/adopted before they were given access, and are left with a final sample of 3,030 individuals.

B Intention-To-Treat Results

In this appendix, we investigate the ITT effects for the Microsoft and Accenture experiments. We first estimate the ITT effect and then conduct a heterogeneity analysis based on ITT estimates.

B.1 ITT Estimates of Productivity Effect

As both experiments feature imperfect compliance because the Control group is eventually granted access to GitHub Copilot—see Figure 2—we drop observations starting just before significant control group adoption. In particular, we restrict attention to the first 29 weeks for the Microsoft experiment and the first 21 weeks for the Accenture experiment. We report the effects of simply being assigned to the treated group (without necessarily adopting Copilot) during those weeks in Table 5. We also report the IV results corresponding to these ITT estimates, noting that they will generally differ from the results reported in Table 9 due to differences in the sample period and weighting.

With the notable exception of the number of builds in the Accenture experiment, we find no statistically significant impact of being assigned to treatment on outcomes. However, we note that adoption rates during these early stages of the experiment were quite small: at Microsoft, only 44.2% of developers adopted Copilot in the first 29 weeks (and this is with adoption significantly accelerating towards the end of this period). At Accenture, adoption was only slightly higher at 61.7% by the end of the initial phase.

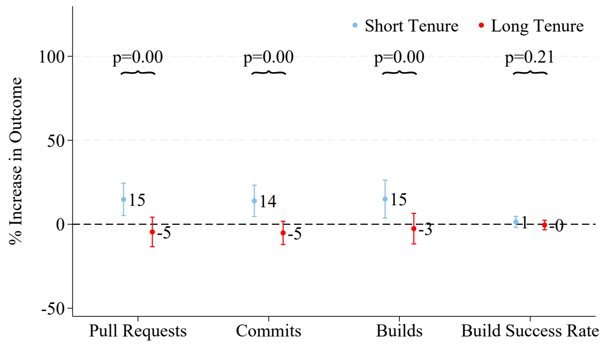
Outcome	Microsoft		Accenture		Pooled	
	ITT	IV	ITT	IV	ITT	IV
Pull Requests	3.91	22.19	9.41	18.65	4.66	20.16
	(3.83)	(22.02)	(9.65)	(18.96)	(3.56)	(14.37)
Commits	3.20	18.19	1.65	3.28	3.08	11.93
	(3.40)	(19.45)	(11.54)	(22.86)	(3.26)	(14.81)
Builds	5.09	28.89	56.45***	111.90***	9.49**	64.84***
	(4.22)	(24.27)	(13.79)	(27.77)	(4.04)	(18.27)
Build Success Rate	0.33	1.81	-9.73*	-17.24*	-0.22	-5.14
	(1.38)	(7.65)	(5.73)	(10.09)	(1.34)	(6.1)

Table 5: ITT Estimates of Effect of Copilot.

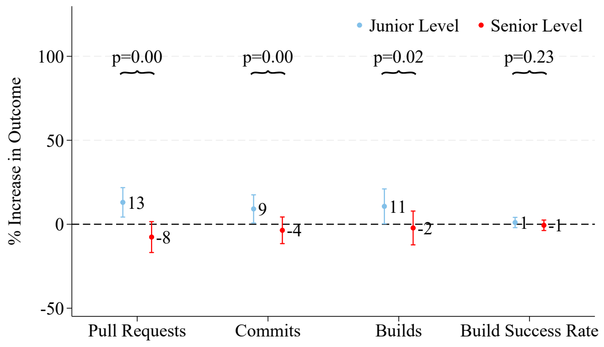
Notes: We investigate the ITT effects of assigning a developer to the treatment group in the Microsoft and Accenture experiments, cutting the sample just before the Control group adopts Copilot. With the exception of builds at Accenture, we find no statistically significant effect of being assigned to the treatment group on outcomes. All effects were estimated in linear regressions but are expressed as a % of the pre-treatment mean. *10%, **5%, ***1%

B.2 Heterogeneity in ITT Estimates

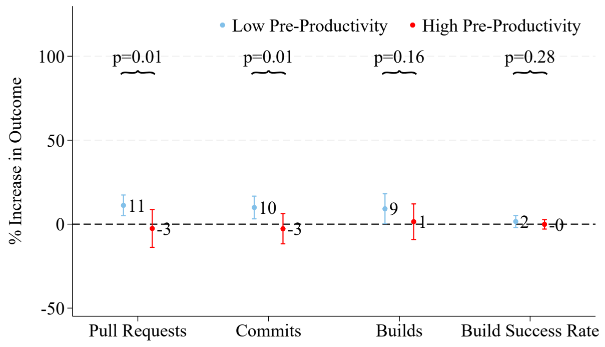
In Figure 7, we explore heterogeneity in the ITT effects, finding that these effects are significantly higher for developers with short tenure, at a junior level, or with low pre-productivity. However, we emphasize that this heterogeneity could be driven by both (i) differences in adoption or utilization and (ii) differences in the effect of Copilot conditional on utilization.



(a) ITT Effects by Tenure



(b) ITT Effects by Level



(c) ITT Effects by Pre-Productivity

Figure 7: Heterogeneity of Copilot Effect (ITT)

Notes: This figure provides ITT estimates of the effect of adopting Copilot on the total number of pull requests, commits, builds, and build success rate broken out by (a) whether a developer's tenure with Microsoft at the beginning of the experiment was below median (short tenure) or above median (long tenure), (b) the level at which a developer was employed and (c) the productivity of the developer before the start of the experiment. The dots in each panel are estimates derived from a single regression for each outcome where the ITT effect is allowed to differ by (a) tenure, (b) level, or (c) the developer's productivity in the pre-period as measured by their total number of completed pull requests. The bars provide 95% confidence intervals based on standard errors clustered at the level of treatment assignment. For all three outcome measures, the ITT effects on productivity are stronger for short-tenure/junior/less productive developers. Note that the ITT effects combine heterogeneity in the amount of uptake with potentially heterogeneous treatment effects on adopters.

C Dynamic Treatment Effects

To explore whether there is any evidence that users have to learn how to use GitHub Copilot, we split our sample into (i) the first three months after adoption, and (ii) more than three months since adoption. We then estimate the DiD model and the weighted IV models, allowing for different effect sizes across these periods (but estimate jointly across periods) in Table 6. The DiD estimates suggest that the treatment effect increases over time, but the experimental IV estimates are underpowered to detect any heterogeneity in effects.

Outcome	DiD		WIV	
	Short Term	Long Term	Short Term	Long Term
Pull Requests	2.16	10.55***	32.39*	31.51
	(2.45)	(3.35)	(18.39)	(22.17)
Commits	4.24*	7.13**	29.33	14.83
	(2.37)	(3.24)	(18.41)	(19.41)
Builds	4.43*	7.46**	36.21	19.55
	(2.57)	(3.62)	(22.21)	(23.58)
Build Success Rate	-1.17	-1.52	-5.47	1.39
	(0.81)	(0.97)	(7.73)	(6.57)

Table 6: Dynamic Treatment Effects

Notes: We separate out treatment effects by whether a developer has adopted GitHub Copilot within the last three months ("Short Term") or has used GitHub Copilot for at least three months already ("Long Term"). While DiD estimates suggest the treatment effect may be growing over time, our experimental (weighted IV) estimates are underpowered to confirm this using experimental variation. All effects were estimated in linear regressions but are expressed as a % of the pre-treatment mean. *10%, **5%, ***1%

D Comparison between Experiments

In this Appendix, we provide additional details on the experiments, emphasizing differences in implementation across companies, the distinct roles of software developers within each firm, and how outcome variables—such as pull requests—should be interpreted across these different contexts. To preserve confidentiality, we cannot describe specific internal details from each organization. However, we offer a qualitative depiction of the differences in developers' day-to-day tasks across the three companies based on publicly available information. We also discuss how differences in the experimental designs may have contributed to heterogeneity in the estimates.

D.1 Design Differences

Although all three field experiments share the same objective, their designs necessarily vary to accommodate each firm's workflows and operational constraints. The Microsoft experiment was launched first, leveraging the company's early access to Copilot as its owner. The experiment ended earlier than planned when control-group developers began to request access mid-experiment. Accenture's experiment, by comparison, started later but lasted for a full five months as planned. The experiment at Accenture also incorporated a training component. The anonymous company took a different approach, implementing a randomized cohort rollout instead of random assignment to a treatment and control group.

Because these experiments were run by the companies in collaboration with Microsoft and we only analyzed the data, they were not pre-registered. Nevertheless, the goal of each experiment is the same across companies: to learn the productivity effects of GitHub Copilot and use that information in their business decisions. Even though the objective is the same, implementation was driven by idiosyncratic constraints, such as the number of software developers who could participate, the duration for which the control group's Copilot access could be restricted, how quickly the firm needed the experiment results, and the cost of the experiment.

There are several ways these constraints vary across companies. For instance, the ability of the experiment coordinator to persuade managers to postpone the rollout of Copilot varies depending on the managers involved and the perceived importance of Copilot. This, in turn, affected the duration of each experiment. For example, in the anonymous company, this constraint led to a randomized staggered rollout rather than a full RCT with treated and control groups. Similarly, at Microsoft, the control group gained access to GitHub Copilot earlier than planned because some teams requested access to it since they were working on AI-related products at the time. Another example is the encouragement design: Microsoft used a simple email, while Accenture implemented comprehensive training sessions. These differences reflect variation in both intervention costs and how firms and their employees perceive them.

D.2 Differences in Software Development

While all three companies in our study employ software developers, these developers differ in their day-to-day tasks and workflows—factors that influence how GitHub Copilot is integrated into their daily work and affect productivity.

At Microsoft, most developers are responsible for maintaining and evolving long-lived first-party products and cloud services. Their code ships directly to users and is updated continuously; therefore, the teams own design, implementation, automated testing, and post-release telemetry in a single, tight feedback loop. Microsoft also employs internal systems that help developers work quickly while maintaining high quality. For example, before any code changes can be added to the main product, they must pass through automated checks and reviews by other team members (gated pull-request policies). The company also uses automated systems that test code and deploy updates (CI/CD pipelines), plus real-time monitoring tools (dashboards) that show how the software is performing.[23]

Accenture is a service-based company that specializes in resolving client issues and providing support for pre-existing applications, primarily to external clients, rather than developing proprietary products. The majority of software development work focuses on client projects across various industries. Developers are assigned to specific client engagements spanning various industries, including banking, insurance, telecommunications, and public services. While Accenture does have some enterprise software products and platforms, these represent a smaller portion of its business. The diverse client base exposes Accenture's software developers to a wide variety of client projects. They might build entirely new software systems, improve existing ones, or fix problems when things break. This means they need to understand the client's business needs, design solutions, write code, and test that everything works properly. Because they work with many different clients, these developers learn multiple programming languages and gain skills across various types of software and computer systems.[24]

Since the manufacturing firm that hosted the third experiment is anonymous, we describe its environment in terms of typical large, hardware-centered manufacturers, rather than providing firm-specific details. In such companies, software developers usually work on three overlapping roles—(i) firmware developers who write code that runs on the computer chips that control hardware components, (ii) factory-automation engineers who build supervisory control software for assembly lines, and (iii) integration engineers who link production equipment with plant-wide IT and quality systems. Each of these categories follows development schedules that combine software updates into planned releases, prioritizing reliability over rapid changes.

We account for these differences between companies in several ways in our empirical analysis. First, throughout the paper, we report percentage changes in the outcome variables rather than levels, since baseline levels are likely to vary across companies. Second, we report estimates for each company separately and only pool them in our mean specification after estimating individual effects. Finally, all of our estimates are based on within-company—and even within-developer—variation, so cross-company differences do not confound the estimates.

[23]Source: Microsoft—How Microsoft Develops DevOps.

[24]Sources: Accenture—Job Details1, Accenture—Job Details2, Accenture—Newsroom Fact Sheet.

D.3 Differences in Outcome Variables

In our baseline results, we used pull requests as the primary outcome variable. Although the primary structure of pull requests remains the same across companies, approaches to pull requests can vary depending on the main product and the company structure. While startups might allow developers to self-merge small changes with minimal review to maintain a rapid pace, established enterprises in regulated industries typically require multiple approvers, comprehensive automated testing, and sometimes security team sign-offs before any code reaches production. The tooling and automation integration also spans a wide spectrum, from companies that run extensive CI/CD pipelines[25] with security scans and performance benchmarks on every PR (code review), to those relying more heavily on manual testing and simpler approval workflows. These differences in merge strategies, branch protection rules, and review culture generally evolve as companies mature, typically moving toward more structured and rigorous processes as team sizes grow and the business impact of code changes increases.

Based on publicly available sources, we provide an overview of Microsoft's internal pull request (PR) practices.[26] Microsoft's PR practices center on Azure DevOps, utilizing a trunk-based workflow. In this approach, developers create short-lived branches off the main branch, and each pull request triggers automated builds and tests to enforce branch policies. Only after these checks pass do team peers review the code; Microsoft requires at least one approval to ensure code quality and architecture standards are met before merging it into the main branch. At Microsoft, pull requests and code reviews are frequent: according to Macleod et al. (2018), 36% of Microsoft developers review code multiple times per day.[27] Such reviews are important in maintaining quality and coordinating tens of thousands of developers on shared codebases.

At Accenture, the most important difference from Microsoft is that pull requests modify a client's live codebase rather than an internal one. This substantially increases the consequences of coding errors. As a result, every PR at Accenture must pass through multiple review and validation stages before approval, making them far less frequent. This is also evident in Table 2, which shows a significantly lower average number of weekly pull requests at Accenture compared to Microsoft (0.13 vs. 0.86).

D.4 The Effects of Company-Level Differences on Results

Because the experiment design choice (timing, training, cohorting) was tailored to specific firm-level constraints, measured treatment effects are inevitably affected by experimental design and company context. Therefore, it is not possible for us to pinpoint the exact sources of heterogeneous effects across these companies. However, we can describe some possible sources of this heterogeneity.

[25]CI/CD stands for Continuous Integration/Continuous Deployment, which refers to automated systems that regularly test, integrate, and deploy code changes to ensure software quality and enable rapid releases.

[26]Sources: Microsoft—How Microsoft Develops DevOps, Microsoft—Transforming Modern Engineering at Microsoft, Greiler—Code Reviews at Microsoft.

[27]MacLeod, Laura, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Cz- erwonka (2018). "Code Reviewing in the Trenches: Challenges and Best Practices". In: IEEE Software 35.4, pp. 34–42.

One potential source arises from the nature of the tasks developers perform in these companies. As described above, coding tasks vary significantly across these companies, inherently generating heterogeneity in the effectiveness of coding assistant tools. For example, coding tools typically perform better on programming languages and tasks that are well-represented in the training data, and some companies may use languages or work on tasks that were more extensively covered in the AI model's training. Although we have limited visibility into the languages used by developers (in some experiments, only for Copilot users), it is plausible that Accenture uses a greater variety of languages and more niche languages than other companies, given their work across multiple industries as described above. This diversity could make Copilot less effective on these less common programming languages, potentially explaining the lower impact observed at Accenture relative to Microsoft.

Another example is selection into treatment. As described in Section 4.4, our analysis identifies the LATE, capturing heterogeneous effects driven by selection into treatment. Selection into treatment can vary across companies due to factors such as company culture and policies. Therefore, even if the underlying productivity effects of coding assistants are identical across companies, differences in how participants select into treatment could lead to heterogeneous results.

Finally, the experimental design can indirectly influence the selection into treatment. For instance, in the Accenture experiment, managers encouraged their direct reports to use the tool to enhance compliance, whereas at Microsoft, a simple email was sent. This difference can affect developers' motivations for adoption: at Accenture, developers might use the tool primarily to satisfy managerial expectations, driving the relatively fast and widespread adoption observed in Figure 2b. At Microsoft, by contrast, adoption is initially slower and less extensive, leaving more scope for developers to select into treatment based on anticipated productivity benefits. This can be a source of a higher LATE estimate at Microsoft than at Accenture. Similarly, the timing of the experiment can affect results. GitHub Copilot likely received updates during the experimental period, potentially leading experiments conducted earlier, such as Microsoft's, to observe a smaller impact.

While we think that the sources of heterogeneity across companies are extremely important, we ultimately have a sample size of three, which prevents us from obtaining systematic evidence on the differences across companies.

E The First Accenture Experiment

We do not discuss in detail in the main text another experiment conducted by Accenture in April 2023, which included a number of Accenture offices located in Southeast Asia. This experiment was abandoned by the company after Accenture laid off 19,000 employees that same month (cnn.com), including 42% of the developers participating in this experiment. Still, this attrition was balanced across treatment and control, and we can thus subset to the 204 developers who were not let go for our analysis; indeed, Table 7 confirms that after this subsetting, treatment and control are still balanced. The problem emerges because Microsoft did not log all Copilot usage data for this experiment, as the company considered it abandoned. In particular, we lack adoption data for the control group until October '23. Without this adoption data, any analysis is potentially biased.

Still, because our initial analysis revealed that this experiment was the only experiment across the three in which we have a negative (though statistically insignificant) point estimate for Copilot's effect on productivity, we proceed to analyze this experiment in this appendix by imputing that nobody in the Control group adopts Copilot until October '23, yielding the adoption path in Figure 8. Thus, in the worst-case scenario, it could be that all the adoptions that we attribute to October 2023 already happened right at the beginning of the experiment. This data quality concern means our treatment effect estimates will be conservative (as we may mistakenly count up to 10% of the control group as non-adopters during half of the sampling period).

Keeping in mind this caveat that our treatment effect estimates are potentially conservative, we report the results from this first Accenture experiment in Table 8. We find a negative point estimate of -39.18% (SE: 36.78%) on the number of tasks completed. Still, this estimate has a high degree of statistical uncertainty, and we note that the estimates for the number of commits 43.04% (SE: 43.04%) and builds 12.33% (SE: 53.60%) are both positive, although not statistically significant.

	Control		Treatment			
	Mean	Std. Dev	Mean	Std. Dev	Difference	p-value
Pull Requests	0.08	0.26	0.09	0.29	0.02	0.38
Commits	6.28	11.24	5.28	10.09	-1.00	0.30
Builds	5.32	10.32	5.23	10.52	-0.09	0.93
Build Success Rate	0.49	0.33	0.50	0.33	0.01	0.60

Table 7: Balance Table for First Accenture Experiment

Notes: This table presents a comparison of pre-experimental outcomes in control and treatment groups in the first Accenture experiment. For each measure, we present its mean and standard deviation in the control group and in the treatment group. We also show the mean difference across these groups and the p-value associated with an underlying test of differences in means. The p-values for the differences are calculated using standard errors clustered at the level of treatment assignment.

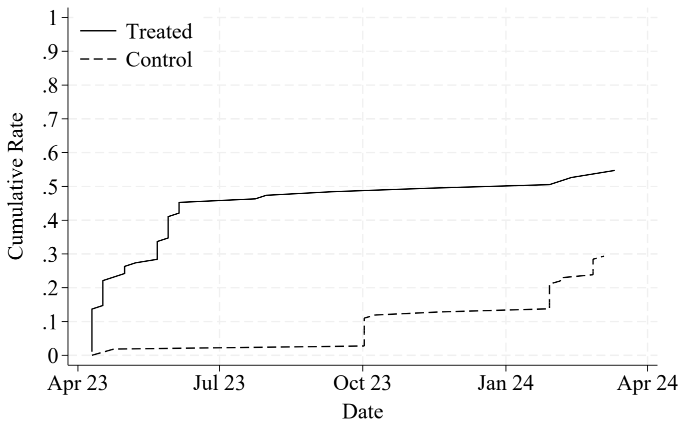
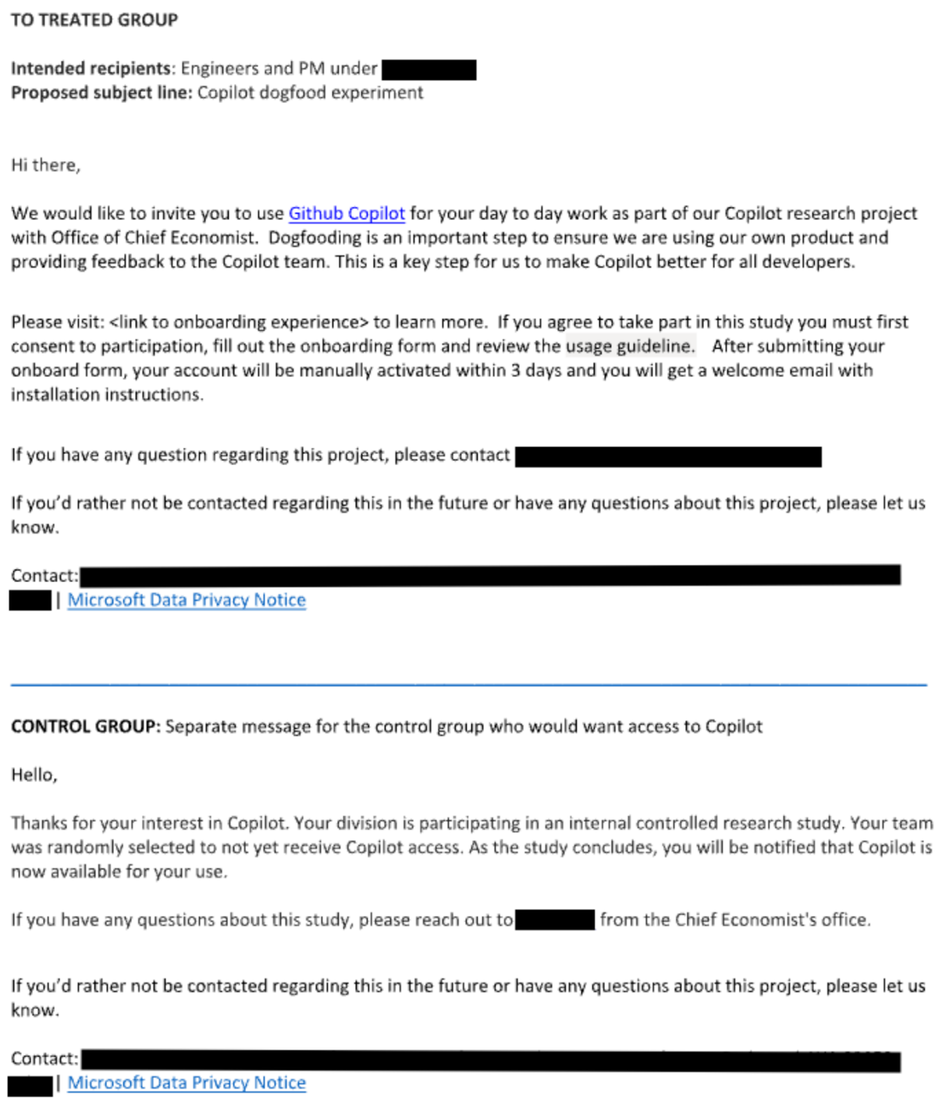


Figure 8: Cumulative Adoption Rates for First Accenture Experiment
Notes: This graph shows the cumulative rate of adoption over time for software developers in both the treatment and control groups in the first Accenture experiment. Note that we are assuming that nobody in the Control group adopts Copilot until October 2023.

Outcome	Accenture #1
Pull Requests	-39.18
	(36.78)
Commits	43.04
	(38.80)
Builds	12.33
	(53.60)
Build Success Rate	-0.99
	(16.51)
N Developers	204
N Clusters	204

Table 8: Weighted IV Results for First Accenture Experiment
Notes: This table provides estimates of the effect of GitHub Copilot adoption on the number of Pull Requests, Commits, Builds, and Build Success Rates in the first Accenture experiment. Standard errors are clustered at the developer level. The estimates presented in this table are potentially conservative because they require imputing that nobody in the Control group adopts Copilot until October 2023.)

F Additional Exhibits & Robustness Checks



TO TREATED GROUP

Intended recipients: Engineers and PM under [redacted]
Proposed subject line: Copilot dogfood experiment

Hi there,

We would like to invite you to use Github Copilot for your day to day work as part of our Copilot research project with Office of Chief Economist. Dogfooding is an important step to ensure we are using our own product and providing feedback to the Copilot team. This is a key step for us to make Copilot better for all developers.

Please visit: <link to onboarding experience> to learn more. If you agree to take part in this study you must first consent to participation, fill out the onboarding form and review the usage guideline. After submitting your onboard form, your account will be manually activated within 3 days and you will get a welcome email with installation instructions.

If you have any question regarding this project, please contact [redacted]

If you'd rather not be contacted regarding this in the future or have any questions about this project, please let us know.

Contact: [redacted]
[redacted] | Microsoft Data Privacy Notice

CONTROL GROUP: Separate message for the control group who would want access to Copilot

Hello,

Thanks for your interest in Copilot. Your division is participating in an internal controlled research study. Your team was randomly selected to not yet receive Copilot access. As the study concludes, you will be notified that Copilot is now available for your use.

If you have any questions about this study, please reach out to [redacted] from the Chief Economist's office.

If you'd rather not be contacted regarding this in the future or have any questions about this project, please let us know.

Contact: [redacted]
[redacted] | Microsoft Data Privacy Notice

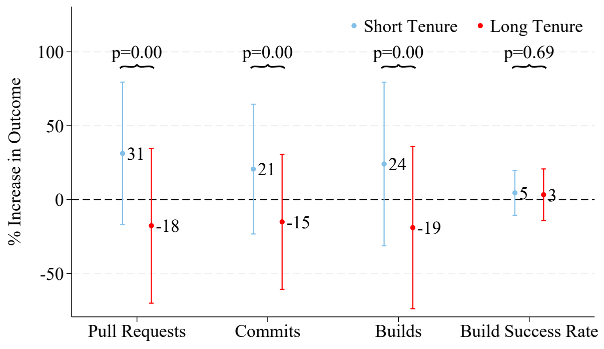
Figure 9: E-mail Sent to Participants in the Microsoft Experiment

Notes: This figure exhibits the copy that was sent to participants in the Microsoft experiment. Only the subset of users in the Control group who explicitly requested access to GitHub Copilot received the second email.

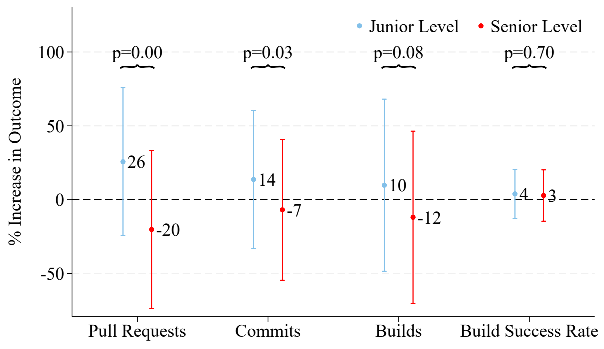
Outcome	Microsoft				Accenture				Anon. Comp.				Pooled			
	DiD	DiD-P	IV	W-IV	DiD	DiD-P	IV	W-IV	DiD	DiD-P	IV	WIV	DiD	DiD-P	IV	W-IV
Pull Requests	7.63***	6.81***	10.53	27.38**	52.65***	22.54**	15.97	17.94	1.70	2.77	54.03	54.03	6.24***	5.23***	18.73	26.08**
	(2.49)	(2.52)	(24.82)	(12.88)	(9.46)	(9.35)	(21.26)	(18.72)	(2.47)	(2.35)	(42.63)	(42.63)	(1.72)	(1.69)	(15.1)	(10.3)
Commits	7.03***	6.42***	5.54	18.32	12.85	-8.67	-3.60	-4.48	-	-	-	-	7.25***	5.82**	0.97	13.55
	(2.32)	(2.46)	(22.20)	(11.25)	(11.62)	(12.08)	(22.19)	(21.88)	-	-	-	-	(2.28)	(2.41)	(15.69)	(10.0)
Builds	7.11***	7.25***	5.87	23.19	39.66***	13.70	96.05***	92.40***	-	-	-	-	8.23***	7.56***	49.66**	38.38***
	(2.65)	(2.74)	(27.25)	(14.20)	(14.03)	(12.10)	(28.05)	(26.78)	-	-	-	-	(2.6)	(2.67)	(19.55)	(12.55)
Build Success	-0.65	-0.66	3.92	-1.34	-20.72***	-19.59***	-18.10*	-17.40**	-	-	-	-	-1.13	-1.05	-5.39	-5.53
Rate	(0.79)	(0.78)	(8.17)	(4.23)	(5.06)	(5.36)	(9.55)	(7.12)	-	-	-	-	(0.78)	(0.77)	(6.21)	(3.64)

Table 9: Alternative Specifications for Experimental Results

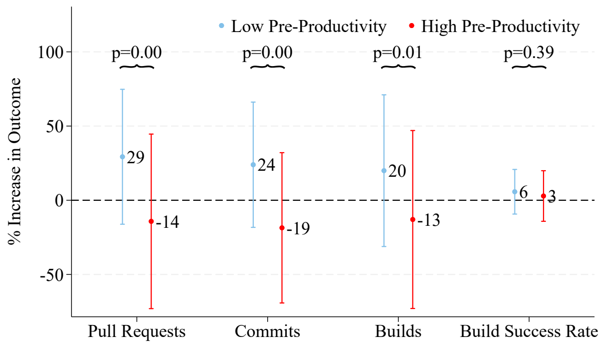
Notes: This table builds on Table 3 by reporting the results of additional specifications. Each entry can be interpreted as an estimate of the percentage effect of adoption of GitHub Copilot. DiD is like in Table 3, DiD-P is like DiD but runs a Poisson model and then reports $100 * (\exp(\beta) - 1)$), IV is like W-IV in Table 3 but without weighting the regression by adoption differences, W-IV is like in Table 3.



(a) Treatment Effects by Tenure



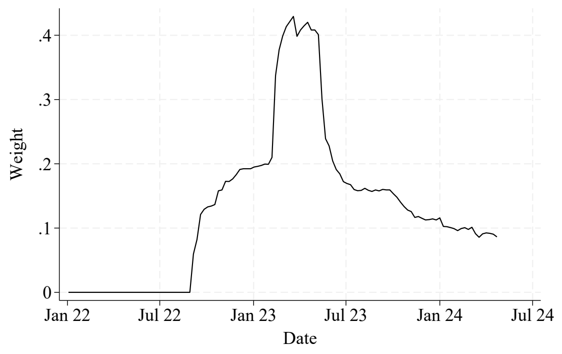
(b) Treatment Effects by Level



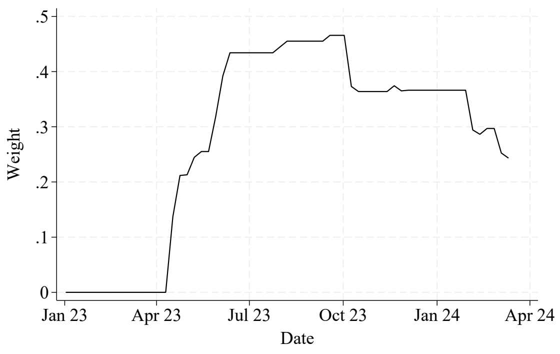
(c) Treatment Effects by Pre-Productivity

Figure 10: Heterogeneity of Effect of Copilot (Unweighted IV)

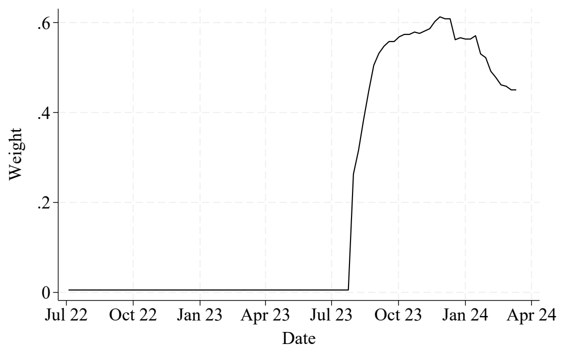
Notes: This figure provides unweighted IV estimates of the effect of adopting Copilot on the total number of pull requests, commits, builds, and build success rate broken out by (a) whether a developer's tenure with Microsoft at the beginning of the experiment was below median (short tenure) or above median (long tenure), (b) which level a developer was employed at, and (c) the developer's pre-experiment productivity. The dots in each panel are estimates derived from a single regression for each outcome where the treatment effect is allowed to differ by (a) tenure, (b) level, or (c) the developer's productivity in the pre-period as measured by his total number of completed pull requests. The bars provide 95% confidence intervals based on standard errors clustered at the level of treatment assignment. For the first three outcome measures, the effects on productivity are stronger for short-tenure/more junior/less productive developers. The p-values for differences between individual coefficient estimates are often very small despite substantial overlap in the confidence intervals due to high correlations (exceeding 0.9) between the estimates.



(a) Microsoft Experiment



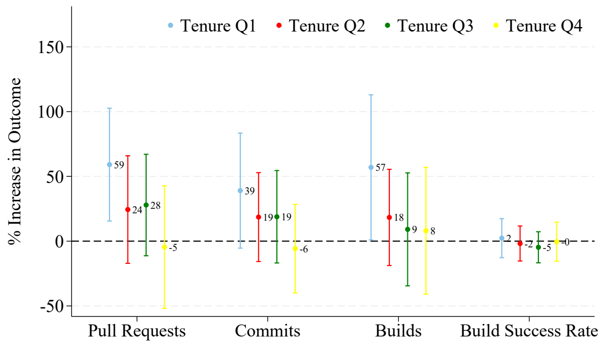
(b) Accenture Experiment #1



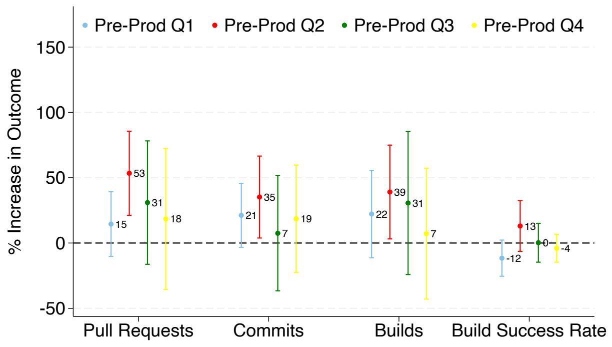
(c) Accenture Experiment #2

Figure 11: Regression Weights

Notes: This figure provides the weights used in the W-IV estimates underlying Tables 3 and 8. Recall that we are weighting the IV estimates to exploit information from periods where the instruments predict uptake. Hence, we use the difference in adoption across the control and treatment groups by a given date as our weight. This matters most for the Microsoft experiment, in which the control group adopted at an elevated rate when its access was granted in March 2023. For this experiment, we put extra weight on the period just before the control group was allowed to adopt Copilot.



(a) Treatment Effects by Tenure



(b) Treatment Effects by Pre-Productivity

Figure 12: Heterogeneity of Copilot Effect based on Quantiles

Notes: This figure provides weighted IV estimates of the effect of adopting Copilot on the total number of pull requests, commits, builds, and build success rate, broken out by (a) the developer's tenure with Microsoft at the beginning of the experiment, and (b) the developer's productivity prior to the start of the experiment. In both cases, developers were grouped into four categories based on quartiles of the corresponding variable. The dots in each panel are estimates derived from a single regression for each outcome where the treatment effect is allowed to differ by (a) tenure, and (b) the developer's productivity in the pre-period as measured by his total number of completed pull requests. The bars provide 95% confidence intervals based on standard errors clustered at the level of treatment assignment. For the first three outcome measures, the effects on productivity are stronger for short-tenure/junior/less productive developers, though the difference is typically not statistically significant.