The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot

Fangchen Song
University of Texas at Austin
2110 Speedway
Austin, Texas
United States
78712
fangchen.song@mccombs.utexas.edu

Ashish Agarwal
University of Texas at Austin
2110 Speedway
Austin, Texas
United States
78712
ashish.agarwal@mccombs.utexas.edu

Wen Wen
University of Texas at Austin
2110 Speedway
Austin, Texas
United States
78712
wen.wen@mccombs.utexas.edu

ONLINE APPENDIX

Appendix A: Estimation Details of the Generalized Synthetic Control Method (GSCM)

Our main causal inference uses the GSCM (Xu 2017). Below, we provide a brief overview of its estimation process for our study.

Model: The GSCM combines the Interactive Fixed Effects (IFE) model (Bai 2009), which considers latent factors with effects that can vary across time and units, with the Synthetic Control (SC) method (Abadie et al. 2015), which creates synthetic control units to act as counterfactuals for the treated units. This combination enables GSCM to (1) relax the assumption of pre-treatment parallel trends, and (2) consider potential unobserved factors that vary over time at the unit level. Furthermore, GSCM offers several advantages over other estimation methods. Unlike the SC method, which is limited to a single treated unit, GSCM is designed for multiple treated units, eliminating the need to construct synthetic control units for each treatment unit one by one. Additionally, unlike the typical difference-in-differences model combined with a specific matching technique, which is most effective for a single treatment turn-on time and a large control group, GSCM allows each treatment unit to have a different treatment period and can efficiently construct synthetic control units from a relatively small control sample. These advantages make GSCM particularly well-suited to our data, as projects adopted Copilot at different points in time and we have relatively fewer projects in the control sample than in the treatment sample.

First, the GSCM adopts a linear IFE framework to model the latent factors. In our study, the outcome Y_{it}, the log number of merged pull requests (PR) and the log average time taken to merge a single PR of project i in month t, can be expressed as:

$$Y_{it} = \delta_{it} D_{it} + X_{it}'\beta + \lambda_i' f_t + \varepsilon_{it} \qquad (1)$$

where D_{it} is the treatment indicator which equals one if project i has been developed with Copilot by month t and zero otherwise; the parameter of primary interest is δ_{it}, which signifies the dynamic impact of Copilot on the log number of merged PRs and the log average time taken to merge a single PR. δ_{it} is the heterogeneous treatment effect and its subscripts i and t indicate that the estimates vary across units and time. Let r be the number of latent factors, then f_t represents the $(r \times 1)$ vector of unobserved common

factors, and λ_i' is the $(1 \times r)$ vector of unknown factor loadings. The factor component $\lambda_i' f_t$ can be expressed as $\lambda_i' f_t = \lambda_{i1} f_{1t} + \lambda_{i2} f_{2t} + \cdots + \lambda_{ir} f_{rt}$. A two-way fixed effects specification is a special case of the factor component, where $r = 2$, $f_{1t} = 1$, and $\lambda_{i2} = 1$, so that $\lambda_i' f_t = \lambda_{i1} + f_{2t}$. Here, λ_{i1} represents the project fixed effects, while f_{2t} is the month fixed effects. We specify the two-way fixed effects to control for heterogeneity across projects and time, while considering other unobserved latent factors. In general, $\lambda_i' f_t$ is estimated by an iterative factor analysis of the residuals from the model. If optimal number of unobserved latent factors determined by the cross-validation technique is zero, it means that the fixed effects setting has effectively accounted for any unobserved time-varying characteristics (Xu 2017). Lastly, ε_{it} is the error term with a mean of zero.

Estimation of Latent Factors: An essential task when utilizing the IFE framework is to identify the number of latent factors, denoted as r. Xu (2017) suggests a predictive analytics method for this purpose. For clarity, we divide the total number of projects into two groups: Tr for treated projects and Co for control projects. The latent-factor selection algorithm first applies Equation (1) to data from control units only, covering both pre-treatment and post-treatment time periods, using the following equation:

$$Y_{it} = X_{it}' \beta + \lambda_i' f_t + \varepsilon_{it}, \forall\, i \in Co \qquad (2)$$

Since the control projects never received treatment during the data-collection period, $\delta_{it}\, D_{it}$ is excluded from Equation (2). The value of r can vary within a range of candidate values specified by the researchers. For each specified r, the algorithm runs Equation (2) and derives the estimates $\hat{\beta}$ and $\widehat{f_t}$. Subsequently, a cross-validation procedure is carried out for all treated projects. Specifically, let s be the index of a pre-treatment period, ranging from one (the first pre-treatment period) to t^0 (the last pre-treatment period). Equation (3) is then applied to each pre-treatment period, starting with $s = 1$, for all projects in the treatment group by leaving one period out and using the remaining periods to estimate the factor loadings.

$$\hat{\lambda}_{i,-s} = (\hat{f}_{-s}^{0\prime}\, \hat{f}_{-s}^{0})^{-1}\, \hat{f}_{-s}^{0\prime}(Y_{i,-s}^{0} - X_{i,-s}^{0\prime}\hat{\beta}),\ i \in Tr,\ s = 1, \ldots, t^0 \qquad (3)$$

In Equation (3), $\hat{f}_{-s}^{0}$ and $\hat{\beta}$ are the estimates obtained from Equation (2). The superscript 0 indicates periods before the introduction of the treatment, and the subscript $-s$ denotes all periods except for s. Based on these estimates, the algorithm predicts the outcome for treated unit i in period s using $\hat{Y}_{is}(0) = X_{is}'\hat{\beta} + \lambda_{i,-s}'\hat{f}_{s}$ and records the out-of-sample error $e_{is} = Y_{is}(0) - \hat{Y}_{is}(0)$ for all $i \in Tr$. In the final step, the algorithm calculates the Mean Squared Error (MSE) of the prediction, summed over all pre-treatment periods s, for each candidate value of r. The value of r that minimizes this prediction error is selected, and the corresponding set of latent factors will be used in the causal inference process.

Estimation of Average Treatment Effects: The GSCM algorithm uses the estimated function to predict the counterfactuals for treated units, denoted as $\hat{Y}_{it}(0)$, representing the outcome of treated projects had they not received treatment in the post-treatment period. The causal effect of the treatment is quantified as the Average Treatment effect on the Treated unit (ATT), calculated mathematically as:

$$ATT_t = \frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}}\left[Y_{it}(1) - \hat{Y}_{it}(0)\right] \text{ for } t > t^0 \qquad (4)$$

where $\mathcal{T}$ denotes the set of treated units and $|\mathcal{T}|$ represents the number of units in $\mathcal{T}$. $Y_{it}(1)$ is the observed outcome for treated project i at time t. The essence of GSCM lies in the fact that if the predicted outcomes for the treated projects during the pre-treatment periods are accurate, the algorithm can produce a valid counterfactual for each treated unit during the post-treatment periods. Finally, the GSCM uses bootstrapping to estimate confidence intervals and standard errors (Xu 2017).

Unlike typical econometric modeling, the latent-factor selection algorithm in GSCM prioritizes models with the lowest out-of-sample prediction error to accurately predict counterfactuals, rather than those with the best fit based on information criterion. As a result, models with more latent factors may not be selected, even if they account for more confounding factors. Moreover, the GSCM without any latent factor still yields a more valid estimate than a difference-in-differences (DID) model, except under two conditions: (1) the treatment is randomly assigned, satisfying the parallel trend assumption; and (2) no treatment heterogeneity exists. If either of these conditions is not met, the DID estimate is likely to be invalid.

Appendix B: Equivalence Test of Pre-trend in GSCM

We apply the equivalence test to examine the presence of a pre-treatment trend in GSCM (Pan and Qiu 2022, Egami and Yamauchi 2023, Wang et al. 2023). This is a standard way to test the performance of GSCM (Liu et al. 2024), as the equivalence test better incorporates substantive considerations of what constitutes good balance on covariates and placebo outcomes compared to traditional tests (Hartman and Hidalgo 2018). Specifically, we use the two-one-sided t (TOST) test. The test includes an equivalence range within which differences are deemed inconsequential (Hartman and Hidalgo 2018, Lakens et al. 2018). The test is considered passed if the average prediction error for any pre-treatment period is within the equivalence range (Liu et al. 2024). Hartman and Hidalgo (2018) recommend calibrating equivalence range based on the standard deviation of the outcome when standardized benchmarks are unavailable. Building on this idea, Liu et al. (2024) propose a benchmark of ±0.36 times the standard deviation of the residualized untreated outcome, and this practice has been adopted in subsequent studies examining topics such as mobile app adoption and investor behavior as well as the impact of ride sharing platforms on public transit systems (Pan and Qiu 2022, Liu et al. 2025). Importantly, the implementation we use follows this logic but allows the threshold to vary with the data and estimation uncertainty. As a result, the implied equivalence range in our setting correspond to approximately 0.14 to 0.26 times the standard deviation of the residualized untreated outcome across different outcomes, which are more conservative than the commonly used 0.36 benchmark.

The result of the equivalence test for the main outcome variables, including project-level code contributions, coordination time and overall project-level timely merge of code contributions which combines the effect of these two competing forces, are shown in Figures B.1, through B.5. We observe that the average prediction error with 90% confidence intervals (the gray dotted line) is within the equivalence range (the red dotted line). Thus, we can reject the null of inequivalence (equivalence test p-value is 0) and conclude that there exists no pre-treatment trend (Hartman and Hidalgo 2018, Pan and Qiu 2022, Egami and Yamauchi 2023, Wang et al. 2023, Liu et al. 2024). This indicates that a sufficient set of confounders have been controlled to address the endogeneity concerns and that GSCM provides a good control group.

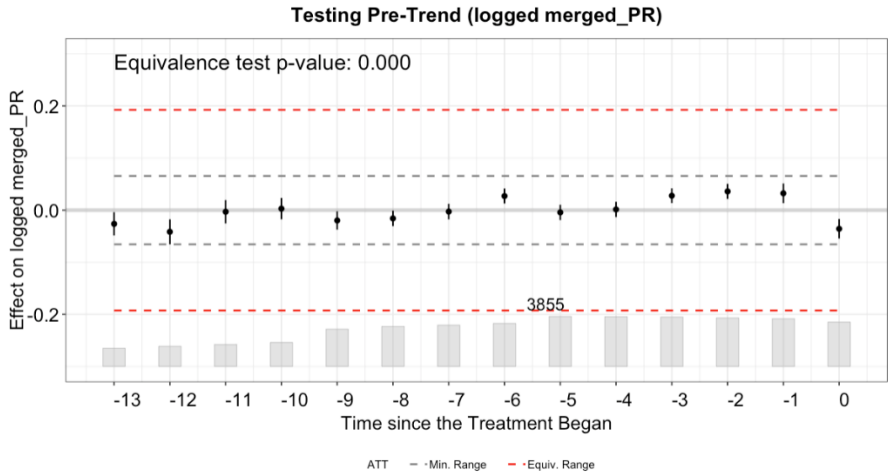

Figure B.1: Pre-trend test (TOST) of logged merged_PR

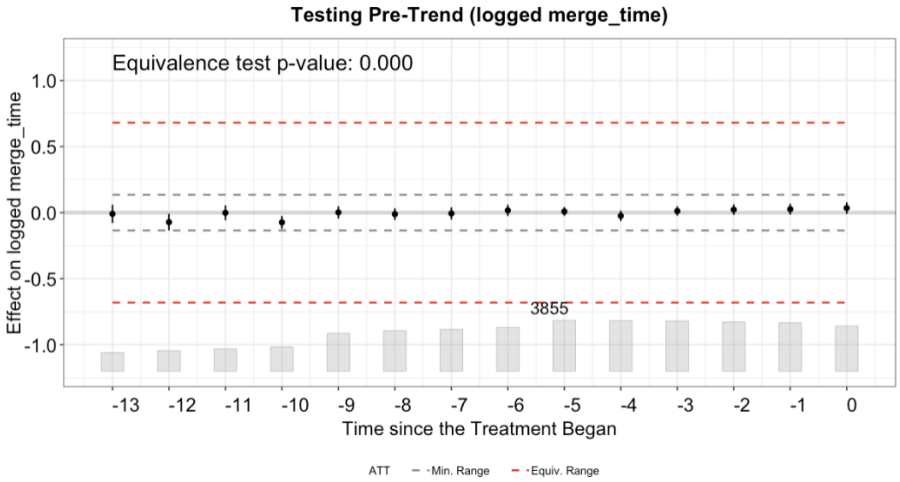

Figure B.2: Pre-trend test (TOST) of logged merge_time

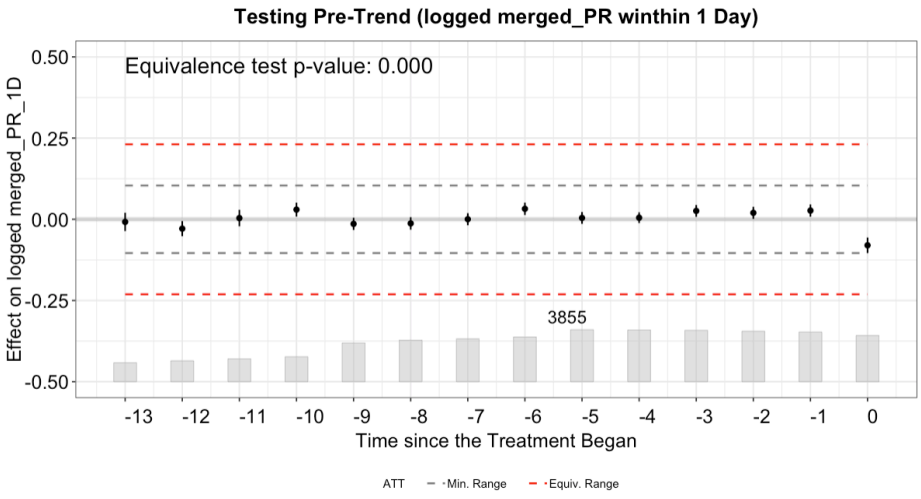

Figure B.3: Pre-trend test (TOST) of logged merged_PR_1D

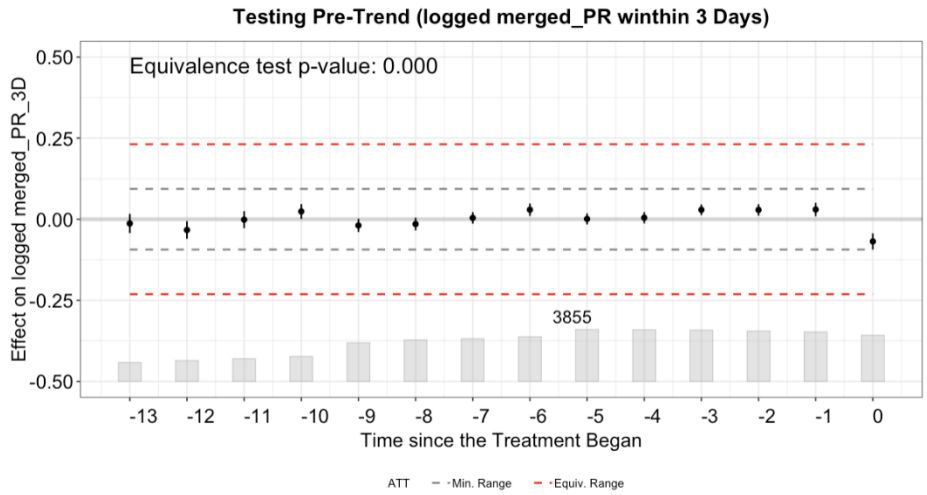

Figure B.4: Pre-trend test (TOST) of logged merged_PR_3D

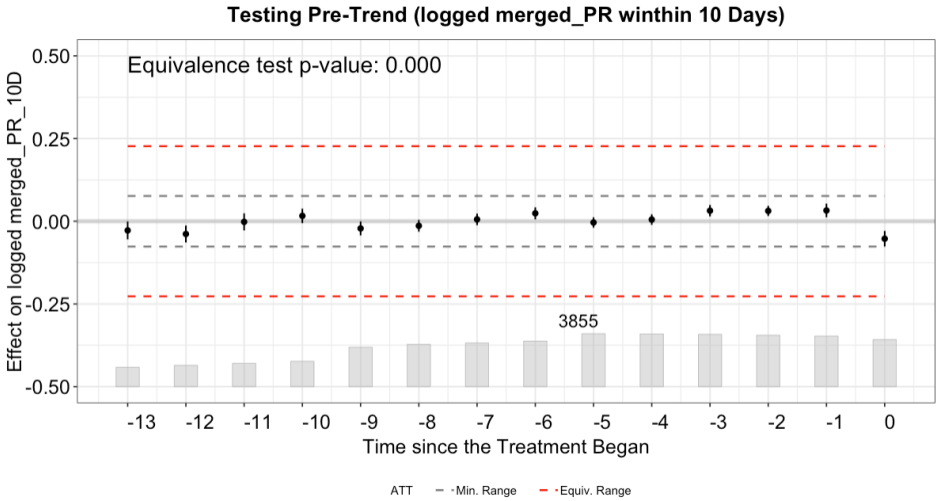

Figure B.5: Pre-trend test (TOST) of logged merged_PR_10D

Appendix C: Placebo Test in GSCM

To further validate our causal identification strategy based on GSCM, we conduct placebo tests by examining pre-treatment periods. Specifically, we define a placebo window covering the three periods prior to treatment and re-estimate the model to assess whether any spurious effects arise before the actual adoption of Copilot. These falsification tests help evaluate the strict exogeneity assumption and detect potential model misspecification. If our identification strategy is valid and the observed effects are driven by Copilot adoption, we should not observe meaningful effects in the pre-treatment period.

To statistically evaluate the results from our placebo analyses, we apply the two-one-sided t (TOST) equivalence test, which assesses whether the estimated placebo effects are practically equivalent to zero (Liu et al. 2024). This approach specifies an equivalence interval around zero and tests the null hypothesis that the placebo effects are meaningfully different from zero.

The results from these tests for the main outcome variables, including project-level code contributions, coordination time and overall project-level timely merge of code contributions, are shown in Figures C.1, through C.5. We observe that the estimated effects in the pre-treatment placebo are at or below zero, while those from the post-treatment period are substantially above zero. The equivalence tests reject the null of inequivalence, indicating that there are no meaningful pre-treatment effects. These findings support the validity and robustness of our causal identification strategy.

It is worth noting that there is a drop at the onset of treatment (i.e., time = 0) in contribution-related measures, such as the number of merged PRs and the number of timely merged PRs. This drop likely reflects an initial adjustment period as developers adapt to the AI pair programmer or integrate it into their workflow. Importantly, the equivalence holds in the pre-treatment window, supporting the credibility of the synthetic control construction.

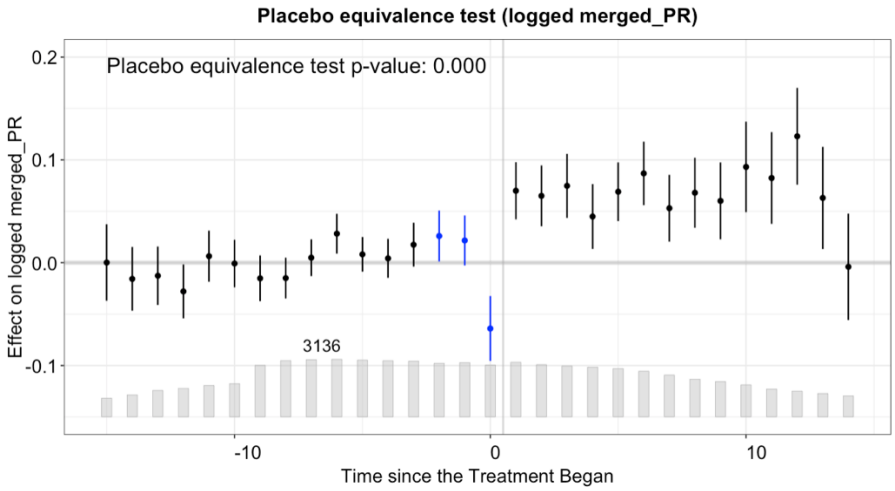

Figure C.1 Placebo Equivalence Test of logged merged_PR

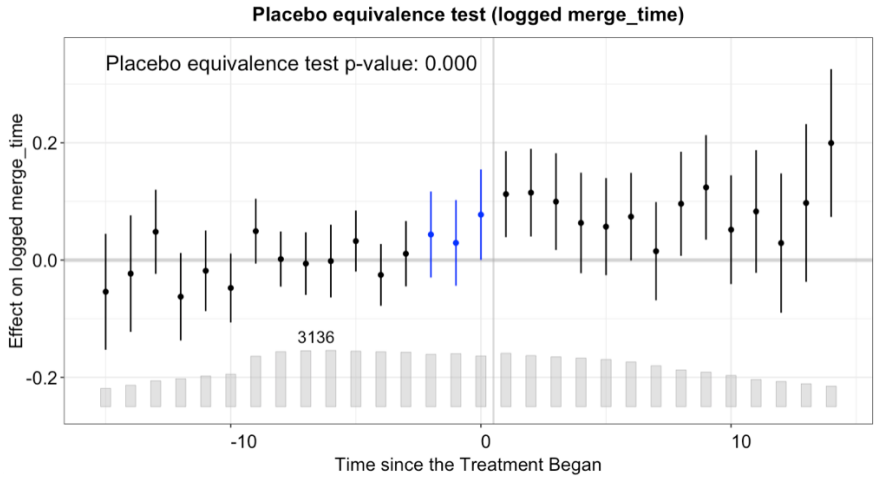

Figure C.2: Placebo Equivalence Test of logged merge_time

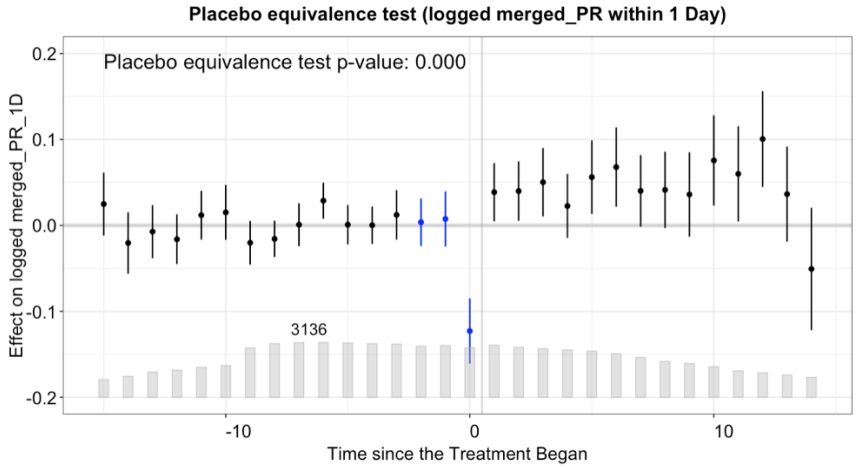

Figure C.3: Placebo Equivalence Test of logged merged_PR_1D

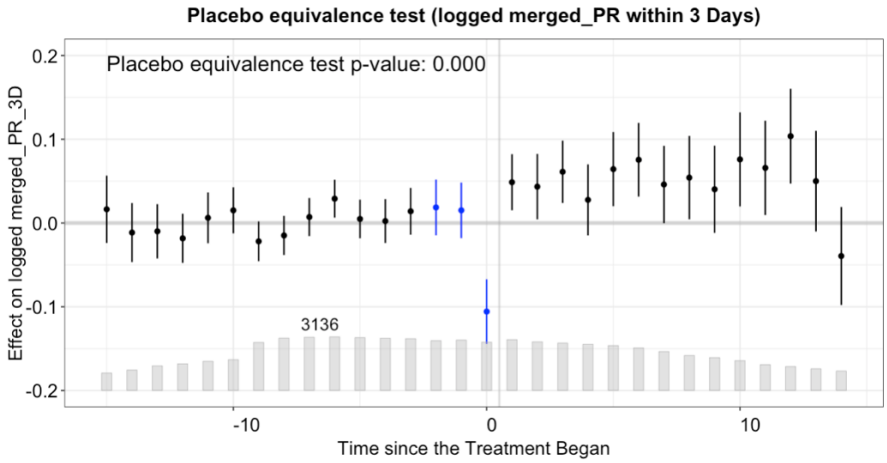

Figure C.4: Placebo Equivalence Test of logged merged_PR_3D

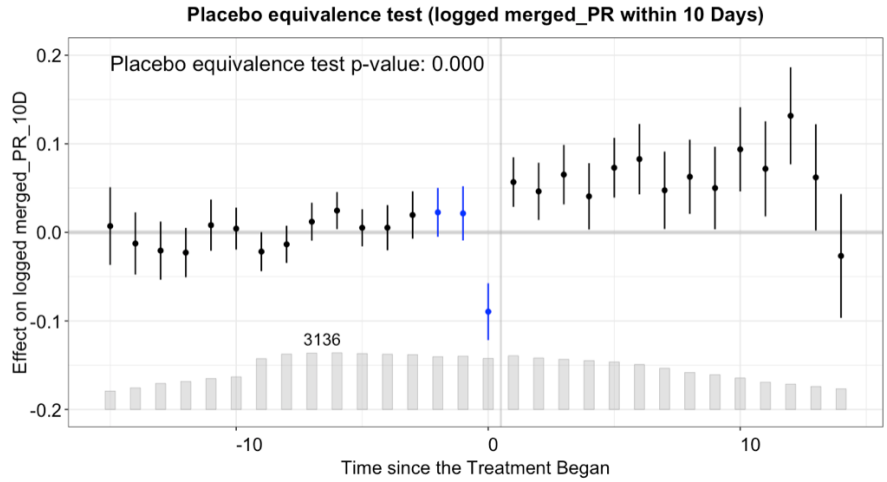

Figure C.5: Placebo Equivalence Test of logged merged_PR_10D

Appendix D: Within-IDE Analysis

In our baseline identification, we use IDEs that supported Copilot during our sample period as part of the criteria to identify projects in the treatment group and consider projects developed with IDEs that did not support Copilot for the control group. To mitigate concerns regarding potential unobservable differences in IDE usage, we conduct the analysis by only focusing on projects using Copilot-supported IDEs and further restricting the treatment versus control group comparison within the same Copilot-supported IDE. This approach addresses three key sources of potential bias. First, it controls for the differences between projects using supported versus unsupported IDEs, the latter of which may be associated with lower popularity. Second, it controls for variation across different supported IDEs, which may attract distinct types of projects or developer teams. Third, it accounts for the possibility that developers who choose IDEs compatible with Copilot may differ from those who do not. By focusing on comparing projects developed within the same Copilot-supported IDE, we ensure that both treatment and control groups share the same development environment and that contributing developers are likely to have similar preferences and characteristics. Therefore, the only variation across groups lies in the timing of Copilot adoption.

Specifically, our first step is to identify the sample of projects using Copilot-supported IDEs. Then, in order to identify treatment and control groups, we use projects that adopted Copilot in the first half of the period from July 2021 to December 2022 when Copilot was available, i.e., July 2021 - March 2022, as the treatment group. Accordingly, we conduct this analysis using only the period from January 2021 to March 2022, so projects that adopted Copilot after March 2022 can be considered as the control group. We choose March 2022 as the endpoint so that there are enough projects with Copilot usage during this time window (i.e., a total of nine months from July 2021 to March 2022). Our second step is to match treatment and control projects within the same IDE. After matching, we use the pooled sample to run the GSCM model. This analysis includes 3,215 projects in the treatment group and 1,247 projects in the control group, a total of 4,462 projects. The treatment turn-on time for each project in the treatment group is decided in the same manner as in our baseline analysis. The results based on estimation using GSCM in Table D.1

show that Copilot increased the number of merged PRs and the average time taken to merge a single PR, which are consistent with our main findings.

Table D.1: The Impact of Copilot on Project-Level Code Contributions and Coordination Time (Within-IDE Analysis)

	(1) Log (Merged_PR)	(2) Log (Merge_time)
ATT of Copilot	0.031***	0.055**
	(0.012)	(0.027)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	4,462	4,462
Observations	60,659	60,659

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. Please refer to Table 1 for detailed definitions of the dependent variables used in this table. *** p<0.01, ** p<0.05, * p<0.1.

Appendix E: PSM and DID

We further validate our results using an alternative matching and estimation approach. Specifically, we use the difference-in-differences (DID) estimation combined with the propensity score matching (PSM). In this analysis, we use June 2021, the first availability date of Copilot, as the single treatment turn-on time for all projects classified into the treatment group in the baseline analysis. Hence, this approach may alleviate concerns about non-random treatment turn-on time for Copilot usage in the treatment group in the main analysis. However, we acknowledge that because developers may sign up gradually after the initial availability date, this approach may consider the period when Copilot was actually not used as the post-treatment period, leading to an underestimation of the positive effect.

We calculate each project's propensity score, defined as the probability of adopting Copilot, using a logit regression model. To construct the covariates for matching, we calculate the monthly values of several project characteristics over the pre-treatment period (January to June 2021), average them across the six-month window, and then apply a log transformation. Specifically, we use the log number of merged PRs, the log average time taken to merge a single PR, the log number of unique developers whose PRs have been merged, the log number of push events, the log number of release events, the log number of opened issues, and the log number of closed issues.

To construct a matched sample, we employ a one-to-one nearest neighbor matching approach without replacement, pairing each treated project that adopted Copilot with a single control project that did not use Copilot. Matching without replacement ensures that each control project is used only once, preventing any single control project from disproportionately influencing the results. While this may reduce the total number of matches, it improves the integrity of the comparison by limiting reuse of controls. Following this matching process between projects where Copilot was used and those that Copilot was not used, we arrive at a dataset comprising 2,122 projects, consisting of 1,061 treatments and 1,061 controls. We report the details of the balance checks of our matched sample in Table E.1. Overall, there are no statistically significant differences between the treatment and control groups across these pre-treatment covariates after matching.

Table E.1: Balance Check of Matched Sample

Variable	Treatment Mean	Control Mean	Difference	P-value
Log (Merged_PR)	2.10	2.05	0.05	0.20
Log (Merge_time)	4.12	4.09	0.03	0.60
Log (Num_devs_with_mergedPR)	1.32	1.29	0.03	0.13
Log (Push)	2.96	2.91	0.05	0.33
Log (Release)	0.28	0.27	0.01	0.79
Log (Open_issue)	1.13	1.11	0.02	0.68
Log (Closed_issue)	1.05	1.04	0.01	0.79

Based on the matched sample, we analyze the impact of Copilot on the number of merged PRs and the average time taken to merge a single PR using the following regression specification:

$$Y_{it} = \beta_0 + \beta_1 Copilot_i \times Post_t + \alpha_i + \mu_t + \varepsilon_{it} \quad (5)$$

where the dependent variable Y_{it} represents the log number of merged PRs and the log average time taken to merge a PR of project i in month t. $Copilot_i$ is a binary indicator that is denoted with "1" for project i where Copilot was used and "0" otherwise. $Post_t$ is a binary indicator that is set to "1" in months after June 2021 and "0" otherwise. To control for time-invariant heterogeneity across projects and time trends, we include project-level fixed-effects α_i and month-level fixed-effects μ_t.

The main coefficient of interest is β_1, as it indicates the change in the number of merged PRs or the change in the average time taken to merge a PR of projects before and after the availability of Copilot, relative to the changes of projects where Copilot was not used at all throughout the sample period. We report the results in Table E.2. The results show that Copilot significantly increased the number of merged PRs and the average time taken to merge a single PR, which are qualitatively consistent with our findings from the main analysis.

Table E.2: The Impact of Copilot on Project-Level Code Contributions and Coordination Time (PSM and DID)

	(1) Log (Merged_PR)	(2) Log (Merge_time)
ATT of Copilot	0.172***	0.148***
	(0.027)	(0.043)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	2,122	2,122
Observations	42,251	42,251

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. Please refer to Table 1 for detailed definitions of the dependent variables used in this table. *** p<0.01, ** p<0.05, * p<0.1.

We next examine whether there is any pre-treatment trend in the analysis of PSM and DID, i.e., whether the outcome variables of projects in the treatment group are different from that in the control group even before the treatment (i.e., the availability of Copilot) happens. The Relative Time Model (RTM) has been used in the economics and information systems literature to validate the pre-treatment parallel trend assumption (Lu et al. 2019, Alyakoob and Rahman 2022). More specifically, we use the following RTM:

$$Y_{it} = \beta_0 + \sum_{\eta=1}^{q} \beta_{1,\eta} \left(Copilot_i \times Post_{t-\eta}\right) + \sum_{\eta=0}^{p} \beta_{2,\eta} \left(Copilot_i \times Post_{t+\eta}\right) + \alpha_i + \mu_t + \varepsilon_{it} \quad (6)$$

In this model, the sum on the right-hand side represents q lead indicators (anticipatory effects) and p lagged indicators (posttreatment effects) where Y_{it} denotes the log number of merged PRs or the log average time taken to merge a single PR for project i in month t. For our analysis of PSM and DID, we set $q = 4$ and $p = 18$. When the pre-treatment trend exists, the lead indicators would be able to predict changes in dependent variables, the log number of merged PRs and the log average time taken to merge a PR. Conversely, if the pre-treatment trend is unlikely to be a concern, the lead indicators would not predict dependent variables (Autor 2003, Angrist and Pischke 2009). The estimation results of the RTM are shown in Table E.3. We observe that the coefficients for the four lead indicators are statistically insignificant, indicating that there does not exist a pre-treatment trend.

Table E.3: Pre-trend Test for PSM and DID Analysis

	(1) PSM and DID Log (Merged PR)	(2) PSM and DID Log (Merge time)
$Copilot_i \times Pre4m$	-0.033	0.037
	(0.034)	(0.088)
$Copilot_i \times Pre3m$	-0.020	0.094
	(0.038)	(0.090)
$Copilot_i \times Pre2m$	-0.050	0.089
	(0.041)	(0.094)
$Copilot_i \times Pre1m$	-0.031	0.100
	(0.041)	(0.094)
$Copilot_i \times Post0m$	0.036	0.165*
	(0.044)	(0.098)
$Copilot_i \times Post1m$	0.068	0.100
	(0.046)	(0.101)
$Copilot_i \times Post2m$	0.097**	0.090
	(0.047)	(0.100)
$Copilot_i \times Post3m$	0.042	0.237**
	(0.046)	(0.116)
$Copilot_i \times Post4m$	0.159***	0.325***
	(0.047)	(0.100)
$Copilot_i \times Post5m+$	0.175***	0.257***
	(0.042)	(0.075)

Notes: Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Appendix F: Instrumental Variable Analysis

To address potential endogeneity concerns on Copilot use, we conduct an instrumental variable estimation. Our instrument exploits variation in developers' exposure to Copilot through their past collaborators outside the focal project. The instrument is constructed by first identifying the historical collaborators of each developer in the focal project based on their collaboration activities in external projects. We then combine these developer level collaborator networks to obtain the set of prior collaborators associated with the focal project. Next, we identify whether any of these prior collaborators were early Copilot adopters. If the focal project is connected to one or more early adopters, we calculate the length of time between the earliest collaborator's Copilot adoption and the focal project's own adoption. Following Sharma et al. (2025), these two components, namely exposure to an early adopter and the corresponding exposure window, are combined into a single project-month level instrument used in Table F.1, denoted as *Early Copilot Adopter Exposure*.

We believe the first component satisfies the relevance condition because prior research shows that individuals learn from peers through social interactions and shared experience (Foster and Rosenzweig 1995, Conley and Udry 2010, Bollinger and Gillingham 2012, Bailey et al. 2022). Learning about new technologies often occurs through observing others' outcomes and updating beliefs about their profitability, especially under uncertainty, and such learning can diffuse through social and professional networks (Foster and Rosenzweig 1995, Conley and Udry 2010). In addition, social interactions can generate spillovers in adoption decisions, where exposure to peers who have adopted a new technology increases the likelihood of one's own adoption (Bollinger and Gillingham 2012, Bailey et al. 2022). Therefore, developers who had past collaborators who were early Copilot users are more likely to be exposed to relevant knowledge and experiences, increasing their likelihood of adoption.

The second component satisfies the relevance condition by capturing the temporal accumulation of social learning. Prior work suggests that the benefits of new technologies grow as knowledge accumulates over time (Foster and Rosenzweig 1995, Conley and Udry 2010). Moreover, peer effects are persistent and can unfold over extended periods, as individuals continue to be influenced by exposure to others' adoption

behaviors (Bollinger and Gillingham 2012, Bailey et al. 2022). Consistent with this evidence, a longer gap between a collaborator's early adoption and the focal project's adoption provides more opportunities for information exchange and observation, thereby increasing the likelihood of Copilot adoption.

Moreover, this instrumental variable meets the exclusion restriction. Early adoption occurs in unrelated projects outside our analytical sample and thus does not directly affect outcomes in the focal project. Instead, its influence operates only through the focal developers' adoption decision. Because the instrument is constructed from historical collaborations external to the focal project, it is unlikely to be correlated with project level outcomes such as merged PRs or average time to merge a single PR.

Specifically, based on our sample, we define early adopters as developers who adopted Copilot between July and September 2021, shortly after its initial release in June 2021. For a focal developer in a focal project between October 2021 and December 2022, we identify their prior collaborators based on the collaboration activities in external projects in 2020. If any of these external collaborators adopted Copilot during the early period (i.e. July to September 2021), the focal developer is considered exposed to early adoption. We then measure the time between the external collaborator's adoption and the focal project's adoption as the exposure window.

We report the results in Table F.1. The first stage shows a significantly positive relationship between the instrumental variable and Copilot use, with an F-statistic of 268.36, indicating strong instrument relevance. The second stage results indicate that Copilot increased both the number of merged PRs and the average time to merge a single PR, consistent with our main findings. Nevertheless, we acknowledge that developers connected to early adopters may differ in other ways, such as network characteristics or exposure to other technologies. Accordingly, we view this instrumental variable analysis as supplementary to our main identification strategy.

Table F.1: The Impact of Copilot on Project-Level Code Contributions and Coordination Time (Instrumental Variable Analysis)

First-stage:	(1) Copilot Use	Second-stage:	(2) Log (merged_PR)	(3) Log (merge_time)
Early Copilot Adopter Exposure	0.091***	ATT of Copilot	0.070***	0.086**
	(0.006)		(0.017)	(0.038)
F-statistics	268.36			
Project FE	Yes	Project FE	Yes	Yes
Month FE	Yes	Month FE	Yes	Yes
# of projects	2,493	# of projects	2,493	2,493
Observations	37,380	Observations	37,380	37,380

Robust standard errors clustered at project-month level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Appendix G: Additional Robustness Checks

G.1 Refined Sample

As developers work on multiple projects, it is possible that some may be working on both treatment and control projects. Approximately 3.5% of the selected projects in our main analysis have developers involved in both groups. Although these developers might not be using Copilot for the control projects due to IDE restrictions, there is potential for knowledge transfer, which could bias our results. To eliminate this bias, we refine our sample to exclusively assess the impact of Copilot on projects where no developer is involved in both groups. As shown in Tables G.1, the results indicate that Copilot increased the number of merged PRs and the average time taken to merge a single PR, which are qualitatively similar to our main results.

There may be spillover effects wherein projects not using Copilot could still benefit if their developers have prior experience with Copilot from other projects. In such cases, our estimates would represent a lower bound on the true impact of Copilot on code contributions and coordination time. The observed increases in code contributions and coordination time would be more pronounced when compared to projects unaffected by such spillover effects.

Table G.1: The Impact of Copilot on Project-Level Code Contributions and Coordination Time (Refined Sample)

	(1) Log (Merged_PR)	(2) Log (Merge_time)
ATT of Copilot	0.062***	0.080***
	(0.007)	(0.022)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	7,369	7,369
Observations	135,731	135,731

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. Please refer to Table 1 for detailed definitions of the dependent variables used in this table. *** p<0.01, ** p<0.05, * p<0.1.

G.2 Outlier Removal

To rule out the possibility that extreme values are disproportionately influencing the results, we conduct an analysis that excludes projects exhibiting unusually high or low values in terms of the number of merged PRs and the average time taken to merge a single PR. By removing these outliers, we ensure that our results

reflect generalizable trends rather than being driven by a few atypical cases. The results, reported in Table G.2, remain consistent with our main findings.

Table G.2: The Impact of Copilot on Project-Level Code Contributions and Coordination Time (Outlier Removal)

	(1) Log (Merged_PR)	(2) Log (Merge_time)
ATT of Copilot	0.060***	0.074***
	(0.008)	(0.022)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	7,557	7,557
Observations	138,123	138,123

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. Please refer to Table 1 for detailed definitions of the dependent variables used in this table. *** p<0.01, ** p<0.05, * p<0.1.

G.3 Non-AI Topic Projects

Another potential concern is that the observed effects may be driven by project-specific enthusiasm associated with certain topics, particularly those related to artificial intelligence. AI-related projects may attract different groups of developers compared to projects focused on other topics, which could confound the estimated effect of Copilot adoption. To address this concern, we conduct a robustness check by excluding projects associated with AI-related topics and re-estimate the models using the remaining sample. This approach allows us to test whether the impact of Copilot is generalizable across a broader range of project topics and not limited to those inherently aligned with AI development. As shown in Table G.3, the results remain consistent with our main findings.

Table G.3: The Impact of Copilot on Project-Level Code Contributions and Coordination Time (Non-AI Topic Projects)

	(1) Log (Merged_PR)	(2) Log (Merge_time)
ATT of Copilot	0.057***	0.088***
	(0.008)	(0.021)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	6,668	6,668
Observations	121,366	121,366

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. Please refer to Table 1 for detailed definitions of the dependent variables used in this table. *** p<0.01, ** p<0.05, * p<0.1.

G.4 Alternative Measures

In the main analysis, we measure project-level code contributions using merged PRs and coordination time using hours between PR submission and acceptance. To ensure our results are not driven by specific measures, we examine alternative measures, including submitted PRs and commits for project-level code contributions, and time intervals in days and minutes for coordination time. Results in Tables G.4.1 and G.4.2 remain consistent, indicating our findings are robust to outcome variable definitions.

Table G.4.1: The Impact of Copilot on Project-Level Code Contributions
(Alternative Measures of Code Contributions)

	(1) GSCM Main Sample Log (Submitted PRs)	(2) GSCM Main Sample Log (Commits)
ATT of Copilot	0.075***	0.107***
	(0.009)	(0.014)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	7,637	7,637
Observations	139,329	139,329

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table G.4.2: The Impact of Copilot on Project-Level Coordination Time
(Alternative Measures of Coordination Time)

	(1) GSCM Main Sample Log (Merge time in days)	(2) GSCM Main Sample Log (Merge time in minutes)
ATT of Copilot	0.049***	0.101***
	(0.016)	(0.032)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	7,637	7,637
Observations	139,329	139,329

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

G.5 Effect of Workload

One important alternative explanation for the observed increase in coordination time for merging a PR following Copilot adoption is that there were simply more PRs that needed to be reviewed, which could lead to some delay. To address this concern, we include the log number of cumulative submitted PRs till the focal month as a control variable to account for potential increases in overall workload when

analyzing coordination time. The results, reported in Tables G.5, are qualitatively similar to our main results for coordination time.

Table G.5: The Impact of Copilot on Project-Level Coordination Time with Code Submission

	(1) GSCM Main Analysis Log (Merge_time)	(2) GSCM Within-IDE Analysis Log (Merge_time)	(3) GSCM Refined Sample Log (Merge_time)	(4) GSCM Outlier Removal Log (Merge_time)	(5) GSCM Non-AI Topic Log (Merge_time)	(6) PSM and DID Single Treatment Turn-on Time Log (Merge_time)
ATT of Copilot	0.079***	0.055**	0.077***	0.072***	0.087***	0.147***
	(0.023)	(0.028)	(0.021)	(0.023)	(0.021)	(0.044)
Log (Open_PR)	0.031	0.041	0.054*	0.021	0.028	0.057
	(0.029)	(0.041)	(0.031)	(0.032)	(0.032)	(0.037)
Project FE	Yes	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes	Yes
# of projects	7,637	4,462	7,369	7,557	6,668	2,122
Observations	139,329	60,659	135,731	138,123	121,366	42,251

Notes: *Open_PR* refers to the total number of cumulative submitted PRs till the focal month and is added to the regressions to control for the effect of workload. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

G.6 Alternative Estimation

We also validate our results using a two-way fixed effects model applied to the full sample. The findings presented in Table G.6 are qualitatively consistent with our main analysis.

Table G.6: The Impact of Copilot on Project-Level Code Contributions and Coordination Time (Two-way Fixed Effects with Full Sample)

	(1) TWFE Project-level Code Contributions	(2) TWFE Coordination Time
	Log (Merged_PR)	Log (Merge_time)
ATT of Copilot	0.029***	0.036**
	(0.008)	(0.018)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	9,244	9,244
Observations	140,084	140,084

Notes: Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

G.7 Expanded Sample

Because our main sample includes projects with three or more developers, we also examine the robustness of our results using an expanded sample that includes projects with two developers. Results are reported in Table G.7 and are qualitatively similar to our main results.

Table G.7: The Impact of Copilot on Project-Level Code Contributions and Coordination Time
(Expanded Sample that Includes Projects with Two Developers)

	(1) GSCM Expanded Sample Project-level Code Contributions	(2) GSCM Expanded Sample Coordination Time
	Log (Merged_PR)	Log (Merge_time)
ATT of Copilot	0.013**	0.083***
	(0.006)	(0.015)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	12,512	12,512
Observations	240,433	240,433

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

G.8 Additional Alternative Sample

In our main analysis, we restrict the sample to project-month observations in which at least one PR was eventually merged. To assess the robustness of our findings, we replicate the analysis by including all project-month observations, regardless of whether any PRs were merged. As shown in Table G.8, the results remain consistent with our main findings. However, it is important to note that we cannot extend this analysis to coordination time as measuring coordination time requires data on PR acceptance time, which is only available when PR is merged.

Table G.8: The Impact of Copilot on Project-Level Code Contributions
(Expanded Sample that Includes Project-month Observations with Zero Merged PR)

	Log (Merged_PR)
ATT of Copilot	0.066***
	(0.006)
Project FE	Yes
Month FE	Yes
# of projects	8,965
Observations	206,897

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Appendix H: Additional Evidence on the Mechanisms for H1 and H2

Table H.1: How Developer Coding Participation and Individual Productivity are Correlated with Project-Level Code Contributions

	(1) Log (Merged_PR)	(2) Log (Merged_PR)
Log (Num_devs_with_mergedPR)	1.213***	
	(0.006)	
Log (MergedPR_per_dev)		1.173***
		(0.006)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	7,637	7,637
Observations	139,329	139,329

Notes: Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table H.2: How Code Discussion is Correlated with Project-Level Coordination Time

	(1) Log (Merge_time)	(2) Log (Merge_time)	(3) Log (Merge_time)
Log (Comment_per_mergedPR)	0.059***		
	(0.004)		
Log (Num_devs_with_comments_per_mergedPR)		0.259***	
		(0.016)	
Log (Comments_per_dev_per_mergedPR)			0.063***
			(0.004)
Project FE	Yes	Yes	Yes
Month FE	Yes	Yes	Yes
# of projects	7,637	7,637	7,637
Observations	139,329	139,329	139,329

Notes: Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table H.3: How Project-Level Code Contributions and Coordination Time are Correlated with Project-Level Timely Merge of Code Contributions

	(1) Log (MergedPR 1D)	(2) Log (MergedPR 1D)	(3) Log (MergedPR 3D)	(4) Log (MergedPR 3D)	(5) Log (MergedPR 10D)	(6) Log (MergedPR 10D)
Log (Merge_time)	-0.152***		-0.144***		-0.119***	
	(0.002)		(0.002)		(0.002)	
Log (Merged_PR)		1.027***		1.058***		1.069***
		(0.003)		(0.002)		(0.002)
Project FE	Yes	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes	Yes
# of projects	7,637	7,637	7,637	7,637	7,637	7,637
Observations	139,329	139,329	139,329	139,329	139,329	139,329

Notes: Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Appendix I: Additional Analyses on Code Discussions among Developers

I.1 Text Analysis of Comment Content

Prior research in team coordination emphasizes that the content of discussions also plays a role in shaping coordination time, as greater diversity in discussion topics can increase the complexity and difficulty of coordination (Hansen et al. 2018). Building on this insight, we argue that the same logic may apply to software development, where developer code discussions serve as a primary mechanism for coordination. In this context, a broader range of topics discussed may introduce additional coordination challenges.

To examine the effect of GitHub Copilot on the diversity of discussion topics within merged PRs, we apply a Latent Dirichlet Allocation (LDA) topic modeling approach. Specifically, we measure both the number of topics discussed per merged PR and the entropy of topic distribution. We adopt the LDA model for three key reasons. First, LDA addresses the data sparsity challenges inherent in traditional approaches such as TF-IDF. Second, the topics and associated keywords generated by LDA are human-interpretable, in contrast to the latent dimensions produced by models like doc2vec. Third, LDA has been widely validated and adopted in prior literature across various research contexts, including scientific publications (Griffiths and Steyvers 2004, Wang and Blei 2011), social media content (Ramage et al. 2010, Weng et al. 2010, Lee et al. 2016, Shin et al. 2020), business descriptions (Shi et al. 2016), and mobile App description (Lee et al. 2020). The key advantage of using LDA lies in its ability to identify latent themes from the data by grouping semantically related keywords into coherent topics, rather than treating each word as an independent feature. We exclude comments that are not in English or are too short to convey meaningful information. After this step, we estimate the topic model using a corpus of 5,404,735 merged PR comments collected from selected projects between 2021 and 2022. This modeling approach provides a multinomial distribution over the inferred topics for each merged PR, enabling us to assess both the number of distinct topics present and the evenness of their distribution.

A critical hyperparameter in LDA modeling is the number of topics to extract. Several methods have been proposed to determine the optimal number, including metrics such as perplexity, topic coherence, and topic diversity. Given our objective of categorizing advice-related content into distinct yet coherent

themes, we rely on both coherence and diversity scores to guide model selection. Based on these criteria, we identify 10 as the optimal number of topics, balancing topic distinctiveness and coherence.

The results, presented in Table I.1, indicate that following the adoption of Copilot use, there was an increase in both the number of discussion topics and the entropy score per merged PR. This suggests that Copilot introduces diverse code discussion content for specific changes in each merged PR, leading to increased coordination time.

Table I.1: The Impact of Copilot on Project-Level Code Discussion Content Per Merged PR

	(1) Entropy_score_per_mergedPR	(2) Topics_count_per_mergedPR
ATT of Copilot	0.017**	0.029**
	(0.007)	(0.014)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	4,009	4,009
Observations	78,636	78,636

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

I.2 Discussion Cycles

In addition to discussion content, we examine the structure of interactions by analyzing the number of discussion cycles per merged PR, defined as the number of iterative rounds of discussion concerning the same proposed changes within a single merged PR. This measure captures the extent to which a merged PR requires repeated iterations before reaching convergence. The results, reported in Table I.2, indicate that Copilot increased the number of discussion cycles per merged PR. Such iterative adjustments extend the time required to reach agreement and finalize integration, thereby increasing overall coordination time.

Table I.2: The Effect of Copilot on Discussion Cycles per Merged PR

	(1) Log (discussion_cycles_per_mergedPR)
ATT of Copilot	0.011**
	(0.005)
Project FE	Yes
Month FE	Yes
# of projects	7,637
Observations	139,329

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Appendix J: The Impact of Copilot and Project Complexity

In OSS communities, complex projects typically contain more features and code modules. Prior work suggests that developers are more likely to contribute to complex projects, as they offer greater status and visibility benefits (Chengalur-Smith et al. 2010, Hann et al. 2013, Medappa and Srivastava 2019). However, these projects also tend to impose greater demands for knowledge sharing, communication, and mutual understanding (Williams 2011, Zimmermann et al. 2018). While generative AI tools enhance developers' capacity to produce code, they do not reduce the inherent complexity of a project. There are two plausible, yet opposing, mechanisms through which project complexity might influence the observed effects of Copilot.

On one hand, complex projects may attract more contributors due to their higher status potential, which can lead to greater project-level code contributions. However, the increased number of contributors also brings a wider range of perspectives and development styles, making it more difficult to reach consensus and resolve conflicting views, thereby increasing coordination time. On the other hand, these projects may present higher expected costs, which could limit participation in both coding and non-coding activities (i.e., code discussion) and reduce coordination time. In either case, it is important to determine whether the effects we observe are simply artifacts of project complexity rather than the result of Copilot adoption. To address this concern, we conduct the analysis by splitting the sample into complex versus simple projects based on the number of lines of code in each project prior to Copilot adoption. We use a median split, classifying projects above the median as complex and those below the median as simple.

The results in Tables J.1 and J.2 show that both simple and complex projects exhibit similar patterns. Copilot adoption is associated with increased project-level code contributions and longer coordination time in both cases. To further examine this, we conduct PSM and DID analyses using an interaction term for project complexity. We define a binary variable "Complexity" that equals 1 if a project's lines of code exceed the median value before the Copilot use and 0 otherwise. As shown in Table J.3, the interaction term is not statistically significant, indicating no difference in Copilot's impact between simple and complex projects.

Table J.1: The Impact of Copilot on Simple Projects

	(1) Project-level Code Contributions	(2) Coordination Time	(3) Timely Project Outputs (1 Day)	(4) Timely Project Outputs (3 Days)	(5) Timely Project Outputs (10 Days)
	Log (Merged_PR)	Log (Merge_time)	Log (MergedPR_1D)	Log (MergedPR_3D)	Log (MergedPR_10D)
ATT of Copilot	0.046***	0.082**	0.025**	0.029***	0.039***
	(0.011)	(0.032)	(0.012)	(0.011)	(0.011)
Project FE	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes
# of projects	3,591	3,591	3,591	3,591	3,591
Observations	54,404	54,404	54,404	54,404	54,404

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table J.2: The Impact of Copilot on Complex Projects

	(1) Project-level Code Contributions	(2) Coordination Time	(3) Timely Project Outputs (1 Day)	(4) Timely Project Outputs (3 Days)	(5) Timely Project Outputs (10 Days)
	Log (Merged_PR)	Log (Merge_time)	Log (MergedPR_1D)	Log (MergedPR_3D)	Log (MergedPR_10D)
ATT of Copilot	0.086***	0.068**	0.065***	0.071***	0.079***
	(0.012)	(0.029)	(0.014)	(0.013)	(0.014)
Project FE	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes
# of projects	4,046	4,046	4,046	4,046	4,046
Observations	78,706	78,706	78,706	78,706	78,706

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table J.3: The Impact of Copilot on Project Complexity (PSM and DID)

	(1) Project-level Code Contributions	(2) Coordination Time
	Log (Merged_PR)	Log (Merge_time)
Copilot	0.142***	0.150***
	(0.034)	(0.057)
Copilot*Complexity	0.059	-0.003
	(0.039)	(0.063)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	2,122	2,122
Observations	42,251	42,251

Notes: Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Appendix K: Additional Analyses on the Effects of Copilot on Core and Peripheral Developers

We construct a set of metrics to capture absolute changes in project-level outcomes for both core and peripheral developers. These metrics span four key dimensions: project-level code contributions, coordination time, project-level timely merge of code contributions, and the underlying mechanisms including developer coding participation, individual productivity, code discussion volume, code discussion participation, and code discussion intensity per developer.

Specifically, for each project-month, we compute the following variables for core and peripheral developers: (1) the number of merged PRs submitted by each group; (2) the average time taken to merge a single PR submitted by each group; (3) the number of PRs submitted by each group that were merged within one, three, and ten days; (4) the number of distinct core and peripheral developers submitting PRs which were eventually merged; (5) the average number of merged PRs per core and peripheral developer; (6) the average number of comments per merged PR submitted by each group; (7) the average number of unique developers participating in code discussions per merged PR submitted by each group; and (8) the average number of comments per developer per merged PR submitted by each group.

We report the results for core developers' project-level code contributions, coordination time, and project-level timely merge of code contributions in Table K.1 and for peripheral developers in Table K.2. Overall, the findings indicate that both core and peripheral developers benefited from the adoption of Copilot, though the magnitude and nature of these benefits varied. For core developers, Copilot led to a 6.1% increase in project-level code contributions, a 6.7% increase in coordination time, and increases in timely integrated code contributions within one, three, and ten days by 4.4%, 4.8%, and 5.8%, respectively. In contrast, peripheral developers experienced a 5.4% increase in project-level code contributions and a 5.3% increase in coordination time. Their timely integrated code contributions increased by 2.3%, 2.4%, and 3.6% within one, three, and ten days, respectively.

The results for the underlying mechanisms, such as developer coding participation, individual code contributions, and code discussion-related metrics, are presented in Table K.3 for core developers and Table K.4 for peripheral developers. As shown in Table K.3, core developers experienced a 5.5% increase in

developer coding participation, a 5.6% increase in individual code contributions, and increases of 4.1%, 1.6%, and 3.6% in code discussion volume, code discussion participation, and code discussion intensity per developer, respectively.

For peripheral developers, as indicated in Table K.4, they experienced a 2.8% increase in developer coding participation, a 2.2% increase in individual code contributions, and increases of 12.6%, 2.3%, and 11.4% in code discussion volume, code discussion participation, and code discussion intensity per developer, respectively.

Table K.1: The Effect of Copilot on Core Developers (Absolute Changes)

	(1) Project-level Code Contributions	(2) Coordination Time	(3) Timely Merged PRs (1 Day)	(4) Timely Merged PRs (3 Days)	(5) Timely Merged PRs (10 Days)
	Log (Core_ merged_PR)	Log (Core_ merge_time)	Log (Core_ mergedPR_1D)	Log (Core_ mergedPR_3D)	Log (Core_ mergedPR_10D)
ATT of Copilot	0.061***	0.067***	0.044***	0.048***	0.058***
	(0.008)	(0.023)	(0.011)	(0.010)	(0.010)
Project FE	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes
# of projects	6,423	6,423	6,423	6,423	6,423
Observations	118,853	118,853	118,853	118,853	118,853

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table K.2: The Effect of Copilot on Peripheral Developers (Absolute Changes)

	(1) Project-level Code Contributions	(2) Coordination Time	(3) Timely Merged PRs (1 Day)	(4) Timely Merged PRs (3 Days)	(5) Timely Merged PRs (10 Days)
	Log (Peri_ merged_PR)	Log (Peri_ merge_time)	Log (Peri_ mergedPR_1D)	Log (Peri_ mergedPR_3D)	Log (Peri_ mergedPR_10D)
ATT of Copilot	0.054***	0.053*	0.023**	0.024**	0.036***
	(0.008)	(0.028)	(0.010)	(0.012)	(0.010)
Project FE	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes
# of projects	4,154	4,154	4,154	4,154	4,154
Observations	77,888	77,888	77,888	77,888	77,888

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table K.3: The Effect of Copilot on Core Developers (Absolute Changes, Mechanism Analyses)

	(1) Coding Participation	(2) Individual Code Contributions	(3) Discussion Volume	(4) Discussion Participation	(5) Individual Discussion Intensity
	Log (Num_core_with_mergedPR)	Log (MergedPR_per_core)	Log (Comment_per_mergedPR)	Log (Num_dev_with_comments_per_mergedPR)	Log (Comments_per_dev_per_mergedPR)
ATT of Copilot	0.055***	0.056***	0.041**	0.016***	0.036**
	(0.007)	(0.011)	(0.019)	(0.005)	(0.017)
Project FE	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes
# of projects	6,423	6,423	6,423	6,423	6,423
Observations	118,853	118,853	118,853	118,853	118,853

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Table K.4: The Effect of Copilot on Peripheral Developers (Absolute Changes, Mechanism Analyses)

	(1) Coding Participation	(2) Individual Code Contributions	(3) Discussion Volume	(4) Discussion Participation	(5) Individual Discussion Intensity
	Log (Num_peri_with_mergedPR)	Log (MergedPR_per_peri)	Log (Comment_per_mergedPR)	Log (Num_dev_with_comments_per_mergedPR)	Log (Comments_per_dev_per_mergedPR)
ATT of Copilot	0.028***	0.022***	0.126***	0.023***	0.114***
	(0.005)	(0.006)	(0.033)	(0.007)	(0.026)
Project FE	Yes	Yes	Yes	Yes	Yes
Month FE	Yes	Yes	Yes	Yes	Yes
# of projects	4,154	4,154	4,154	4,154	4,154
Observations	77,888	77,888	77,888	77,888	77,888

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

In addition, we examine whether the increase in participation is more pronounced among new or existing peripheral developers. We classify peripheral developers as new or existing based on whether they have previously contributed to the focal project. Following the ratio measure used in the main analyses, we calculate the proportion of new peripheral developers relative to the total number of peripheral developers and determine the effect of Copilot on this variable. A positive estimate indicates a greater increase in the participation of new peripheral developers, compared to existing peripheral developers.

As shown in column (1) of Table K.5 below, there was a significant increase in the share of new peripheral developers following the use of Copilot. Moreover, column (2) of Table K.5 shows a significant increase in new peripheral developer participation as well, suggesting that the observed increase in the share of new peripheral developers was not driven by a decrease in existing developer participation.

Overall, this pattern seems consistent with our cost–benefit argument. For peripheral developers, the expected benefits of contributing to the focal project are relatively stable, as their involvement is typically limited in scope and influence. The introduction of AI pair programmers primarily affects the cost side of the participation decision for them. For new peripheral developers, AI substantially lowers expected costs, thereby increasing the net value of contribution and encouraging participation. In contrast, existing peripheral developers appear to experience smaller cost reductions. With similar expected benefits, the more modest cost savings translate into a relatively limited increase in net contribution value, which weakens their incentive to further increase participation.

Table K.5: The Effect of Copilot on New and Existing Peripheral Developers Participation

	(1) New_dev_with_mergedPR_pct	(2) Log (nums_new_dev_with_mergedPR)
ATT of Copilot	0.039***	0.094***
	(0.003)	(0.007)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	3,584	3,584
Observations	84,807	84,807

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Appendix L: Project Familiarity

We argue that one potential reason that explains the difference in code contributions and coordination time between core and peripheral developers lies in their different levels of project familiarity. In the OSS community, core developers are more embedded in focal projects and maintain sustained involvement (Setia et al. 2012), while peripheral developers contribute more sporadically (Howison and Crowston 2014, Krishnamurthy et al. 2016), which leads to their limited level of project familiarity.

To explore this potential explanation, we first compare the level of project familiarity for core versus peripheral developers. Based on the sample of projects with continuous activity from 2018 to 2021, we measure a developer's project familiarity based on their activity during this period related to a focal project. More specifically, a developer's project familiarity with a focal project is measured by the developer's working tenure with the project, i.e., the number of months a developer contributed to the focal project before June 2021 (i.e., the introduction of GitHub Copilot). As reported in Table L.1, the t-test results show that on average, core developers have significantly longer working tenure than peripheral developers, consistent with our argument that core developers have greater project familiarity.

Table L.1: Comparison between Core and Peripheral Developers on their Project Familiarity, Measured by Project Working Tenure

Variable	Core Developers	Peripheral Developers	Mean diff	p-value
Avg_tenure	6.98	1.64	5.34	0.00

Notes: The table is based on the same sample as the one used in our main analyses in the paper.

We next explore whether the differential effects of Copilot on code contributions and coordination time among core versus peripheral developers can be partly explained by the different levels of project familiarity. To do so, our main idea is to examine whether Copilot had differential effects on code contributions and coordination time for developers with different levels of familiarity. Developers are classified into high versus low project familiarity based on whether their working tenure exceeds the median value of working tenure for that project before the adoption of Copilot. Using the same empirical approach as Section 6.5, we compute the ratio of code contributions made by high familiarity developers, denoted as the *Ratio of merged PR*. We also compute the coordination time required to merge their contributions

relative to the project average, denoted as the *Ratio of merge time*. The results are shown in Tables L.2. Consistent with our arguments, Copilot adoption led to a relatively greater increase in the share of code contributions made by developers with high project familiarity, alongside a relatively greater reduction in the coordination time required to merge their contributions compared to the project level average.

Table L.2: Heterogeneous Effect of Copilot on Developers with High vs. Low Project Familiarity

	(1) Project-level Code Contributions	(2) Overall Coordination Time
	Ratio of merged PR	Ratio of merge time
ATT of Copilot	0.024***	-0.052***
	(0.004)	(0.009)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	6,171	6,171
Observations	87,048	87,048

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

Furthermore, we analyze the content of discussions surrounding both core and peripheral developers' code contributions. We find that discussions related to peripheral developers' contributions frequently focus on clarifying design intent, aligning code with project specific context and historical development, and resolving uncertainties related to integration risks. Such comments are consistent with more limited project familiarity and contextual knowledge, which in turn leads to more discussions during code integration. We provide representative qualitative examples below to illustrate these patterns:

"You can find the table of valid contexts here."

"That's probably better (more semantically accurate) in the context of this project."

""configuring" got me a bit confused, since it sounds like a process, I couldn't figure out which point in time exactly it is."

"I think this changes the value of the global variable XXX if it succeeds"

"I added more context -- let's discuss more here. This is a very risky area of the code for compatibility."

"I'm not sure I do understand the intended flow here. My understanding is XXX. If this is the case, you should rather do something like XXX."

"We would like to avoid using XXX in all other places. So, how about we change this as below?"

To assess whether these qualitative examples are representative, we analyze the content of PR discussions. Specifically, we employ a dictionary-based text analysis approach, which is particularly feasible given the scale of our data, comprising 5,404,735 discussion records. We apply keyword matching to PR comments associated with code contributions by core and peripheral developers. Specifically, we capture context clarification discussion, including comments aimed at clarifying design intent, aligning code with project specific context and historical development, and resolving integration related uncertainties. Prior work documents that such clarification-oriented feedback is more prevalent when contributors have more limited project familiarity (Rullani and Haefliger 2013, Tsay et al. 2014, Joblin et al. 2017).

We calculate the proportion of comments received by core and peripheral developers that relate to context clarification and report these results in Table L.3. The results show that peripheral developers receive a higher proportion of comments related to context clarification, which suggests that they are less familiar with the project context.

Overall, this set of evidence provides some additional support that project familiarity might partially explain the difference in the impact of Copilot on coordination time between core developers and peripheral developers.

Table L.3: Comparison between Core and Peripheral Developers on their Received Comment

Variable	Core Developers	Peripheral Developers	Mean diff	p-value
Context_comment_pct	0.087	0.204	-0.117	0.00

Appendix M: Alternative Explanations for the Differential Impacts between Core and Peripheral Developers

To address the concern that the heterogeneous effects of Copilot on core versus peripheral developers may be driven by differences such as programming language expertise or Copilot usage, we conduct the following additional analyses.

First, it is worth noting that the theoretical distinction between core and peripheral developers does not necessarily reflect differences in programming language expertise. As suggested by the literature, peripheral developers include core developers from other projects and end users (Setia et al. 2012), which indicates that peripheral developers may also possess substantial expertise in the relevant programming languages. Core developers may be new to a domain and rely on peripheral contributors with specialized knowledge to support project development. Alternatively, core developers may be domain experts, while peripheral developers participate in the project to learn and gain experience.

Given the plausibility of both scenarios, we compare the fraction of expert developers in core developers against the fraction of expert developers in peripheral developers. Specifically, we classify developers as programming language experts if they used that language in at least 75% of their prior projects. We then compare each developer's expertise to the primary language used in the focal project to determine whether they met the expert criteria.

As reported in the first row in Table M.1, the t-test result shows no significant difference between core and peripheral developers in terms of the percentage of expert developers. This suggests that differences in language expertise may not be the primary reason for the observed heterogeneity between core and peripheral contributors.

Second, we consider the possibility that core developers benefit more simply because they are more likely to use Copilot or use it more intensively. To explore this possibility, we leverage the proprietary data provided by GitHub organization that indicates for each project, the percentage of Copilot users among core developers and the percentage of Copilot users among peripheral developers. Due to privacy

constraints, we do not have access to individual-level Copilot usage data for specific core or peripheral developers.

As reported in the second row in Table M.1, interestingly, we find that peripheral developers had a significantly higher Copilot adoption rate than core developers. Despite this, our analysis in Section 6.5 of the main paper shows that peripheral developers experienced relatively smaller project-level code contributions and greater coordination time from Copilot usage. This pattern further supports our argument that the observed differences in outcomes are not driven by differential Copilot adoption rates but are more plausibly attributed to differences in project familiarity.

Table M.1: Comparison between Core and Peripheral Developers

Variable	Core Developers	Peripheral Developers	Mean diff	p-value
% of expert developers	0.68	0.67	0.01	0.15
% of Copilot users	0.20	0.30	-0.10	0.00

Notes: The table is based on the same sample as the one used in our main analyses in the paper.

To further address the concern that increased activity by core developers may crowd out contributions from peripheral developers, leading to the cannibalization effect, we include additional controls that capture prior coordination time within the project. Specifically, we control for the lagged log average merge time of core developers' code and the lagged log average merge time of peripheral developers' code. As reported in Table M.2, the results remain consistent after including the controls. That is, we continue to observe an increase in the number of peripheral contributions and a decline in the proportion of peripheral contributions relative to total contributions following Copilot adoption.

Table M.2: The Effect of Copilot on Peripheral Developers' Code Contributions

	(1)	(2)
	Log (Peri_merged_PR)	Peri_merged_PR_share
ATT of Copilot	0.081***	-0.022***
	(0.018)	(0.007)
Log (avg_core_merge_time_lag)	-0.009**	-0.006***
	(0.004)	(0.002)
Log (avg_peri_merge_time_lag)	-0.0002	0.004**
	(0.004)	(0.002)
Project FE	Yes	Yes
Month FE	Yes	Yes
# of projects	4,154	4,154
Observations	77,888	77,888

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

To address the review priority explanation that peripheral developers may wait longer for their code to be reviewed (Yu et al. 2015), we measure coordination time as the duration from the first review to final acceptance, thereby excluding any waiting time prior to the initial review. As shown in Table M.3, both core and peripheral developers experienced an increase in this review duration, consistent with the rise in code discussion reported in Tables K.3 and K.4 of Online Appendix K. Importantly, peripheral developers exhibited a larger increase in this duration. Overall, the results mitigate concerns about review priority, as this alternative measure isolates coordination during the review process rather than delays in obtaining an initial review.

Table M.3: The Effect of Copilot on Review Duration for Core and Peripheral Developers

	(1) Core Developers' Review Duration	(2) Peripheral Developers' Review Duration	(3) Peripheral Developers' Review Duration Relative to Core Developers
	Log (Core_review_to_merge)	Log (Peri_review_to_merge)	Peri_review_to_merge_ratio
ATT of Copilot	0.074***	0.115***	0.049***
	(0.015)	(0.014)	(0.016)
Project FE	Yes	Yes	Yes
Month FE	Yes	Yes	Yes
# of projects	6,423	4,154	4,154
Observations	118,853	77,888	77,888

Notes: All estimations are based on the GSCM method. Robust standard errors clustered at project level in parentheses. *** p<0.01, ** p<0.05, * p<0.1.

References

Abadie A, Diamond A, Hainmueller J (2015) Comparative politics and the synthetic control method. *American Journal of Political Science* 59(2):495-510.

Alyakoob M, Rahman MS (2022) Shared prosperity (or lack thereof) in the sharing economy. *Information Systems Research* 33(2):638-658.

Angrist JD, Pischke J-S (2009) *Mostly harmless econometrics: An empiricist's companion* (Princeton university press).

Autor DH (2003) Outsourcing at will: The contribution of unjust dismissal doctrine to the growth of employment outsourcing. *Journal of labor economics* 21(1):1-42.

Bai J (2009) Panel data models with interactive fixed effects. *Econometrica* 77(4):1229-1279.

Bailey M, Johnston D, Kuchler T, Stroebel J, Wong A (2022) Peer effects in product adoption. *American Economic Journal: Applied Economics* 14(3):488-526.

Bollinger B, Gillingham K (2012) Peer effects in the diffusion of solar photovoltaic panels. *Marketing Science* 31(6):900-912.

Chengalur-Smith I, Sidorova A, Daniel SL (2010) Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems* 11(11):5.

Conley TG, Udry CR (2010) Learning about a new technology: Pineapple in Ghana. *American economic review* 100(1):35-69.

Egami N, Yamauchi S (2023) Using multiple pretreatment periods to improve difference-in-differences and staggered adoption designs. *Political Analysis* 31(2):195-212.
Foster AD, Rosenzweig MR (1995) Learning by doing and learning from others: Human capital and technical change in agriculture. *Journal of political Economy* 103(6):1176-1209.
Griffiths TL, Steyvers M (2004) Finding scientific topics. *Proceedings of the National academy of Sciences* 101(suppl_1):5228-5235.
Hann I-H, Roberts JA, Slaughter SA (2013) All are not equal: An examination of the economic returns to different forms of participation in open source software communities. *Information Systems Research* 24(3):520-538.
Hansen S, McMahon M, Prat A (2018) Transparency and deliberation within the FOMC: A computational linguistics approach. *The Quarterly Journal of Economics* 133(2):801-870.
Hartman E, Hidalgo FD (2018) An equivalence approach to balance and placebo tests. *American Journal of Political Science* 62(4):1000-1013.
Howison J, Crowston K (2014) Collaboration through open superposition: A theory of the open source way. *Mis Quarterly* 38(1):29-50.
Joblin M, Apel S, Hunsen C, Mauerer W (2017) Classifying developers into core and peripheral: An empirical study on count and network metrics. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)* (IEEE), 164-174.
Krishnamurthy R, Jacob V, Radhakrishnan S, Dogan K (2016) Peripheral developer participation in open source projects: an empirical analysis. *ACM Transactions on Management Information Systems (TMIS)* 6(4):1-31.
Lakens D, Scheel AM, Isager PM (2018) Equivalence testing for psychological research: A tutorial. *Advances in methods and practices in psychological science* 1(2):259-269.
Lee GM, Qiu L, Whinston AB (2016) A friend like me: Modeling network formation in a location-based social network. *Journal of Management Information Systems* 33(4):1008-1033.
Lee GM, He S, Lee J, Whinston AB (2020) Matching mobile applications for cross-promotion. *Information Systems Research* 31(3):865-891.
Liu C-W, Mithas S, Pan Y, Hsieh JP-A (2025) Mobile apps, trading behaviors, and portfolio performance: Evidence from a quasi-experiment in China. *Information Systems Research* 36(2):828-846.
Liu L, Wang Y, Xu Y (2024) A practical guide to counterfactual estimators for causal inference with time-series cross-sectional data. *American Journal of Political Science* 68(1):160-176.
Lu Y, Gupta A, Ketter W, Van Heck E (2019) Information transparency in business-to-business auction markets: The role of winner identity disclosure. *Management Science* 65(9):4261-4279.
Medappa PK, Srivastava SC (2019) Does superposition influence the success of FLOSS projects? An examination of open-source software development by organizations and individuals. *Information Systems Research* 30(3):764-786.
Pan Y, Qiu L (2022) How ride-sharing is shaping public transit system: A counterfactual estimator approach. *Production and Operations Management* 31(3):906-927.
Ramage D, Dumais S, Liebling D (2010) Characterizing microblogs with topic models. *Proceedings of the international AAAI conference on web and social media*, 130-137.
Rullani F, Haefliger S (2013) The periphery on stage: The intra-organizational dynamics in online communities of creation. *Research Policy* 42(4):941-953.
Setia P, Rajagopalan B, Sambamurthy V, Calantone R (2012) How peripheral developers contribute to open-source software development. *Information Systems Research* 23(1):144-163.
Sharma V, Agarwal A, Barua A (2025) Demand-side effects of open innovation: The case of cryptocurrency forking. *Management Science* 71(8):7136-7160.
Shi Z, Lee GM, Whinston AB (2016) Toward a better measure of business proximity. *MIS quarterly* 40(4):1035-1056.
Shin D, He S, Lee GM, Whinston AB, Cetintas S, Lee K-C (2020) *Enhancing social media analysis with visual data analytics: A deep learning approach* (SSRN Amsterdam, The Netherlands).

Tsay J, Dabbish L, Herbsleb J (2014) Let's talk about it: evaluating contributions through discussion in GitHub. *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, 144-154.

Wang C, Blei DM (2011) Collaborative topic modeling for recommending scientific articles. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 448-456.

Wang Q, Qiu L, Xu W (2023) Informal Payments and Doctor Engagement in an Online Health Community: An Empirical Investigation Using Generalized Synthetic Control. *Information Systems Research*.

Weng J, Lim E-P, Jiang J, He Q (2010) Twitterrank: finding topic-sensitive influential twitterers. *Proceedings of the third ACM international conference on Web search and data mining*, 261-270.

Williams C (2011) Client–vendor knowledge transfer in IS offshore outsourcing: insights from a survey of Indian software engineers. *Information Systems Journal* 21(4):335-356.

Xu Y (2017) Generalized synthetic control method: Causal inference with interactive fixed effects models. *Political Analysis* 25(1):57-76.

Yu Y, Wang H, Filkov V, Devanbu P, Vasilescu B (2015) Wait for it: Determinants of pull request evaluation latency on github. *2015 IEEE/ACM 12th working conference on mining software repositories* (IEEE), 367-371.

Zimmermann A, Oshri I, Lioliou E, Gerbasi A (2018) Sourcing in or out: Implications for social capital and knowledge sharing. *The Journal of Strategic Information Systems* 27(1):82-100.