

The Impact of Large Language Models on Open-source Innovation: Evidence from GitHub Copilot

Doron Yeverechyahu
Coller School of Management, Tel Aviv University

Raveesh Mayya
Stern School of Business, New York University

Gal Oestreicher-Singer
Coller School of Management, Tel Aviv University

Updated: 16th September 2024
([Click Here](#) for the Latest Version)

Abstract

Generative AI (GenAI) has been shown to enhance individual productivity in a guided setting. While it is also likely to transform processes in a collaborative work setting, it is unclear what trajectory this transformation will follow. Collaborative environment is characterized by a blend of *origination* tasks that involve building something from scratch and *iteration* tasks that involve refining on others' work. *Whether* GenAI affects these two aspects of collaborative work and to *what extent* is an open empirical question. We study this question within the open-source development landscape, a prime example of collaborative innovation, where contributions are voluntary and unguided. Specifically, we focus on the launch of GitHub Copilot in October 2021 and leverage a natural experiment in which GitHub Copilot (a programming-focused LLM) selectively rolled out support for Python, but not for R. We observe a significant jump in overall contributions, suggesting that GenAI effectively augments collaborative innovation in an unguided setting. Interestingly, Copilot's launch increased maintenance-related contributions, which are mostly *iterative* tasks involving building on others' work, significantly more than code-development contributions, which are mostly *origination* tasks involving standalone contributions. This disparity was exacerbated in active projects with extensive coding activity, raising concerns that, as GenAI models improve to accommodate richer context, the gap between *origination* and *iterative* solutions may widen. We discuss practical and policy implications to incentivize high-value innovative solutions.

Keywords: Generative AI, Open-Source Community, Innovation, Productivity, Large Language Models

JEL Classification: O31, C88, J24, O35, L86

1. Introduction

Rapid advances in Generative Artificial Intelligence (GenAI) technologies are promising to transform various aspects of economic activities. Burgeoning research has explored how GenAI can affect the future of work and labor markets (Brynjolfsson et al., 2018; Eloundou et al., 2024), productivity and creativity (Brynjolfsson et al., 2023; Noy and Zhang 2023; Zhou and Lee 2024, Doshi and Houser 2024), and knowledge dissemination (Burtch et al., 2024; Horton 2023; Quinn and Gutt 2023).

An intriguing gap within the burgeoning literature is the lack of empirical investigation into GenAI’s impact on the collaborative work environment—a setting where multiple contributors make a variety of contributions to different aspects of a project. Most of what people do in a collaborative setting often involves a blend of *origination* tasks where solutions are conceptualized and executed from scratch and *iterative* tasks, where solutions are built on top of existing work by others.¹ While *GenAI* can improve individual productivity in guided settings (Brynjolfsson et al. 2023; Noy and Zhang 2023) and has shown mixed outcomes on individual creativity (Chen and Chan, 2023; Doshi and Houser, 2024; Zhou and Lee, 2024), a clear intuition of GenAI’s impact on collaborative tasks is missing. The divergent findings on individual tasks suggest that the impact of GenAI on collaborative innovation are likely to be complex.

This research adopts a unique empirical approach to provide key insights into the nature of GenAI’s effects on collaborative innovation. Our approach is motivated by the intuition that GenAI is likely to excel in *iterative* tasks that build upon others’ work because it assists in understanding the context of other’s work better, compared to *origination* tasks where tasks need to be conceptualized and executed from scratch. To excel in *iterative* tasks, one needs to understand the solution that other have provided in prior iterations and refine it in the current iteration. Such tasks benefit from “interpolative” thinking (i.e., *inside-the-box* thinking), which involves finding solutions within existing knowledge inside the task. To excel in *origination* tasks, one needs to conceptualize and architect the solutions from scratch without a precedence

¹ While the terms *iteration* and *origination* are themselves derived from software development lifecycle literature including the agile development literature (e.g., Curtis et al., 1988, Larman 2004), the definitions easily generalize to all collaborative tasks in knowledge economy, such as customer support, legal advice, creative work, etc.

in the task. Such tasks benefit from “extrapolative” thinking (i.e., *out-of-the-box* thinking), which involves generating novel solutions with no precedence within the task. Figure 1 demonstrates the two types of tasks using an example of plotting regression coefficients. Panel (A) represents an iterative task where the contributor is modifying the values of the background color, line color and the dot size to enhance the readability of the plot when someone prints it using a monochrome printer. Panel (B) represents an origination task as the contributor has to figure out how to compute confidence intervals from existing data as well as to overlay them as vertical lines on top of point estimates.



Figure 1. Illustration of Iterative and Origination Tasks

Note: In the iterative task, the line color, dot size and background color is altered to enhance the readability of the extant plotting function. In the origination task, new functions are written to first compute the confidence interval and then to overlay vertical lines on top of the coefficients.

The intuition that GenAI will have an asymmetric impact on tasks needing *out-of-the-box* thinking versus those needing *inside-the-box* thinking stems from two key observations: (a) most AI systems are evaluated on tasks with well-defined solutions rather than open-ended exploration, and (b) GenAI models predict the most likely response to an input sequence based on their exposure to vast amounts of text,

resulting in outputs that likely reflect established patterns and solutions (Brown et al. 2020). Accordingly, *iterative* tasks—which benefit from detailed context and often have a clear “right answer” or a clearly defined evaluation metric (e.g., drug repurposing or recommender system)—are likely to be well-suited for GenAI. *Origination* tasks, however, involve inherently open-ended or ambiguous contexts and may not have a set of “right answers” (e.g., evaluating the benefits of gene editing technologies), making them less amenable to GenAI. Thus, we propose that GenAI's influence on collaborative innovation is likely intertwined with the balance of *iterative* and *origination* tasks within that domain.

The open-source innovation setting, characterized by its transparent and non-guided problem-solving approach (Lerner and Tirole 2002, Fershtman and Gandal 2011, Belenzon and Schankerman 2008, Medappa et al. 2023) provides an ideal context for testing our ideas about the impact of GenAI on collaborative innovation. The relevance of this setting is further underscored by the increasing adoption of GenAI tools, particularly Large Language Models (LLMs), by software developers across various programming tasks.² As a specific form of collaborative innovation, open-source is driven by “contributors”, typically expert programmers, who volunteer their time to solve problems ranging from innovative architectural solutions to routine implementation solutions. This diversity of cognitive demands, coupled with the voluntary and non-guided nature of contributions (Medappa et al. 2023), allows us to examine how the introduction of GenAI differentially impacts various aspects of innovation.

Within the broader landscape of open-source projects, programming languages like Python, JavaScript, and R, stand as a major pillar of open-source movement with publicly accessible code repositories³. The significant potential of innovation for these languages resides within their extensive ecosystems of user-contributed libraries. These libraries, also known as “packages”, are collections of pre-written programs that expand the capabilities of a language and are maintained by sub-communities of problem solvers. To illustrate, popular Python libraries include Pandas (for data handling) and NumPy (for scientific

² [The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot](#) – GitHub blog

³ Publicly accessible code repository allows anyone to view, modify, and distribute the code, fostering a collaborative and innovative development environment.

computing), while notable R libraries include `ggplot2` (for data visualization) and `caret` (for machine learning). The availability of these libraries, continuously developed and refined by their vibrant communities, not only enhances the utility of these open-source languages (Fershtman and Gandal 2011) but also serves as a crucible for much of their innovative potential. Consequently, our research investigates GitHub Copilot’s impact on open-source innovation at the library/package level.

First, extant research documents the impact of LLMs on individual productivity, especially on low-skilled workers in a guided setting (Noy and Zhang 2024, Peng et al. 2023). However, the influence of LLM within collaborative settings, where contributions intertwine and build upon each other, remains uncharted. Thus, our initial research question examines: *How do LLMs affect the volume of contributions in open-source innovation?* This question is especially pertinent to open-source collaboration. The interconnected nature of contributions in open-source projects, where contributors often build upon other contributors’ work, and the high skill level of open-source contributors, present unique challenges in assessing the impact of LLMs. The voluntary nature of contributions further adds to the complex dynamic of collaborating with other unpaid contributors (Maruping et al. 2019; Von Krogh et al. 2012). In short, the self-selection of high-skilled experts adds to the uncertainty surrounding the impact of LLMs.

Next, building on our aforementioned intuition that certain tasks may be more or less suited to LLM augmentation, we ask: *How do LLMs affect the types of contributions in open-source innovation?* In routine tasks such as writing or customer service, where solutions rely on past examples, LLMs can improve outcomes (Brynjolfsson et al. 2023, Noy and Zhang 2023). However, in less-structured domains that involve creating products from scratch, such as story writing, visual art, or ad copy, LLM’s impact is less straightforward (Chen and Chan 2023, Doshi and Houser 2024, Zhou and Lee 2023). The extent to which we expect a similar phenomenon in open-source collaboration is unclear, given its unique nature. As we will show in what follows, open-source innovation comprises both *iterative* tasks (e.g., maintenance tasks such as bug fixes) and *origination* tasks (e.g., code development tasks such as algorithm implementation). However, given the collaborative and self-selective nature of open-source projects, the effect of LLMs on different types of contributions in open-source innovation is ultimately an empirical question.

Finally, we investigate how the observed effects might evolve as LLMs improve⁴, examining the heterogenous impact across projects with varying levels of user activity. Active projects, characterized by extensive user discussions and rich documentation, inherently facilitate clearer problem definition, which in turn allows for the creation of a more detailed context for LLM usage. Studying the impact of LLMs on active projects allows us to examine how improved LLM context understanding might affect the evolution of open-source contributions. We are particularly interested in assessing whether the gap between iterative and origination tasks narrows, widens or remains stable under improved LLM context understanding. The answer to this question could reveal whether the observed patterns are a transient artifact of current LLM limitations or a more fundamental dynamic in the relationship between GenAI and open-source innovation.

We study these questions in the context of the launch of GitHub Copilot, a widely adopted programming focused LLM⁵. Launched in October 2021, GitHub Copilot initially supported languages like Python while omitting support for languages like R. Both Python and R are prominent open-source data-oriented languages whose packages (e.g., NumPy, Pandas in Python, ggplot2, caret in R) are maintained by thriving communities on GitHub, the leading collaborative software development platform. We leverage these conditions as a natural experiment to compare innovation trajectories on GitHub with and without an LLM.

Our data consists of all activities on 1610 Python and R packages, each tracked over a timeframe from October 2019 to December 2022, including their respective package updates and commits (over 622,000 commits). To analyze the type of innovation, we used the commit comments and leveraged the language discernment capability of LLMs. We benchmarked a few LLMs and ultimately used GPT 3.5 Turbo to classify all GitHub commits into the five categories: Maintenance, Code Development, Documentation and Style, Testing and Quality Assurance, and other non-programming tasks (details in Appendix B).

As our main identification strategy, we employ the differences-in-differences framework, specifically the classical Two-Way Fixed Effect (TWFE) model. This strategy involves Propensity Score Matching

⁴ We assume that improvements in LLMs primarily involve improving context understanding (through expanded windows, better training, or enhanced architectures).

⁵ Over one million developers and 20,000 organizations have adopted GitHub Copilot as of June 2023 - [GitHub Blog](#).

with the nearest neighbor matching, resulting in a dataset comprising 1610 unique Python packages as the treatment group and 1610 R packages as the control group. As an alternative identification strategy, we use *Synthetic Difference in Differences* (SDiD), a modern cutting-edge estimation technique (Arkhangelsky et al., 2021), which builds upon the Synthetic Control technique (Abadie et al., 2010) to non-parametrically ensure the parallel trends assumption required for estimating the impact of GitHub Copilot.

Our analyses uncover some interesting insights. We find that the number of updates for Python packages increases by 17.82% compared to the matched R packages. This increase is driven by a 51% rise in voluntary commits, suggesting that LLMs enhance programmers' voluntary contributions. Regarding the type of innovation, while both the code development and maintenance commits increase significantly, the impact is disproportionately higher for maintenance commits, i.e., about 15.14% higher than code development commits, which increase by 16.76%. This suggests that GitHub Copilot aids iterative tasks requiring interpolative thinking *inside-the-box* more than it aids origination tasks requiring extrapolative thinking *out-of-the-box*. In other words, although LLMs enhance all types of solutions, we document a shift in the trajectory of innovation towards iterative tasks.

The investigation into the distribution of contributions based on user activity footprint reveals that new contributions are disproportionately skewed towards packages with more user activity. Analyzing the relationship between popularity and types of innovation, while code development commits are higher for packages with above-median user activity, maintenance commits show a disproportionately greater increase. Specifically, the increase in *maintenance* commits *viz-a-viz* *code development* commits was 22.01% in more popular projects, compared to the increase in less-active projects (at 8.32%). This finding is important because it suggests that LLM capabilities, being better suited for iterative tasks, may encourage exploitation (maintenance and refinement) over exploration (experimentation and novel development), particularly in projects with abundant contextual resources. This finding has significant implications for open-source platforms like GitHub: while LLMs enhance overall innovation, they may not promote knowledge spillovers from active to niche packages.

We make several contributions to the burgeoning literature on LLMs and open-source innovation. Our study adds to the growing literature on the impact of GenAI on innovation. To the best of our knowledge, we are the first to leverage a unique natural experiment by examining the launch of GitHub Copilot, which supported Python but not in R, to demonstrate that LLMs also significantly impact high-skilled workers and can boost voluntary innovation. This is a departure from previous studies that focused on low-skilled workers in organizational settings where contributions are guided (Brynjolfsson et al. 2023, Noy and Zhang 2023). Our finding that LLMs have a disproportionate impact on maintenance-related commits compared with code development commits has implications for the discourse on the usefulness of LLM solutions in collaborative projects. GitHub Copilot can understand the context of the code, mostly written by others, and offer guidance on iterative refinement. This contextual understanding helps alleviate cognitive burdens associated with identifying flaws in others' logic and subsequent bug fixing without introducing new errors. This scenario inherently allows for a detailed context that enhances the usability of LLMs. Consequently, LLMs significantly improve the efficiency and accuracy of maintenance tasks, particularly in projects with extensive user activities where contextual information is abundant. One key generalization is that LLMs are highly effective in iterative tasks requiring *within-the-box* solutions, such as understanding customer concerns, verifying the accuracy of the accounting data, or retrieving semi-related documents from millions of legal documents. However, LLMs may have a smaller, though potentially valuable, impact on origination tasks that require architectural thinking, such as the design of complex systems. These tasks require higher-level abstract thinking, placing them outside the well-defined solution spaces that LLMs handle best.

2. Literature Survey

2.1 Early Impact of GenAI on Productivity and Creativity

Several well-run experiments provide compelling evidence that GenAI positively impacts individual productivity in guided tasks. For example, Brynjolfsson et al. (2023) examined the staggered deployment of a chat assistant using data from over 5,100 agents at a Fortune 500 software firm that provides business process software. They noted that productivity, measured as issues resolved per hour, increased by 14% on

average, with novice or low-skilled workers improving by 34%. In a similar vein, Noy and Zhang (2023) conducted a preregistered online experiment with 453 college-educated professionals randomly assigned to complete writing tasks with or without access to ChatGPT. They found that ChatGPT substantially boosted average productivity: time taken fell by about 40% and the output quality increased by 18%.

The impact of GenAI on individual creativity is more nuanced. While Doshi and Houser (2024) found that access to GenAI led to stories being evaluated as more creative, they also noted that these stories exhibited greater similarity and reduced diversity, suggesting a potential trade-off. Zhou and Lee (2024) further supported this trade-off, noting that, although GenAI increased the quantity of creative output, it also led to a decline in average content novelty. Chen and Chan (2023) found that using LLMs as “ghostwriters” offered no significant benefits and was even detrimental to expert users, highlighting the potential negative impacts of over-reliance on GenAI for creative tasks. In summary, while GenAI has shown promise in enhancing certain dimensions of individual creativity, there are also potential unintended consequences in other dimensions.

An intriguing gap in the literature is the lack of empirical investigation into the impact of GenAI on collaborative tasks. None of the extant studies have examined how GenAI affects collaborative innovation that intricately mixes both productivity and creativity in unguided settings. Considering that collaborative innovation is a critical component of economic growth and societal development, our study provides valuable insights into this under-explored area.

2.2 Interpolative and Extrapolative Solutions

The tasks and its solution-space can be broadly categorized into two distinct types: *iterative* tasks needing interpolative solutions and *origination* tasks needing extrapolative solutions (Schockaert and Prade 2013).

Interpolative solutions are derived from existing knowledge and patterns. For example, solutions to tasks such as customer service and routine writing often involve searching through past solutions to address present issues (Brynjolfsson et al. 2023, Noy and Zhang 2023). This method relies on interpolative or *within-the-box* thinking, where the task is largely about finding the closest match to the present issue from

previously encountered scenarios. The more detailed the available context, the easier it becomes to generate accurate and relevant solutions. Such tasks usually have an “oracle,” a system that would instantly note what an acceptable solution would look like. In customer service, for instance, agents are expected to analyze past successful interactions and provide relevant responses to current customer concerns. In financial advising, agents are trained to understand consumers’ risk preferences and their existing investment patterns to offer tailored investment recommendations based on successful strategies that have worked for others in the past.

Extrapolative, or *out-of-the-box* solutions involve generating novel approaches and engaging in creative problem-solving (Schockaert and Prade 2013). These tasks are inherently less likely to have a detailed context because merely reviewing past solutions and generating similar responses may not adequately address the issue at hand. For instance, developing a new feature or implementing a novel algorithm requires extrapolative thinking that extends beyond previously encountered scenarios. Evaluating the outcome of a modified gene in a gene editing technology requires extrapolative reasoning, as the edited gene deviates from the path of natural selection without clear indications of whether this intervention represents the optimal path forward or carries unforeseen consequences for the modified gene and its ecology. In such cases, providing more context may not necessarily enhance the usefulness of LLM outcomes (Vaithilingam et al. 2022). Open-source software development is an ideal setting for studying this question because it involves a mix of iterative and origination tasks that require interpolative and extrapolative thinking, respectively. The rapid evolution of open-source projects compared to closed-source projects (e.g., Paulson et al. 2004) further makes the impact of LLMs on these two types of tasks an open empirical question.

3. Research Context and Data

3.1 Empirical Context: The Natural Experiment

Our investigation contrasts the community contributions to open-source packages in Python and R, the two prominent open-source data-oriented programming languages, each significantly influencing various programming fields. They are both cultivated by robust, active communities, a factor that contributes

significantly to their increasing popularity and ensures their status as cutting-edge tools in the technological landscape. These languages are supported by a wealth of libraries and frameworks (hereinafter referred to as *packages*), such as Python’s NumPy and R’s ggplot2, which offer a broad range of sophisticated functionalities suited to a variety of data wrangling and analyzing requirements. The evolution of these *packages* over the years, most of which are open-source projects, highlights how open-source languages are consistently evolving to meet the diverse demands of the programming world.

The open-source communities of packages for both these languages are on GitHub, a platform widely used for collaborative software development. GitHub plays a central role in the ecosystem of open-source projects, with over 100 million software developers contributing on GitHub in over 500 programming languages.^{6 7} On GitHub, these communities are organized as code “repositories”, where contributors can propose and implement code changes through “commits.” Each commit on GitHub is a record of a modification to the source code, capturing the specific changes made, along with metadata such as the author’s name, a timestamp, and a message describing the purpose of the change. While the process allows for meticulous tracking and management of the development process, it allows researchers to investigate the repositories’ evolution through community contributions.

The natural experiment that our investigation exploits is the launch of GitHub Copilot, an AI-powered LLM that provides coding assistance. Specifically, the natural experiment we leverage is the selective support that GitHub Copilot provided, supporting languages such as Python, while not supporting languages such as R. GitHub Copilot was developed collaboratively by GitHub and OpenAI and was unveiled in October 2021. It was designed to enhance coding efficiency across multiple programming languages by offering code snippet suggestions and auto-completing program lines based on the provided context. From its official launch in October 2021, GitHub Copilot demonstrated proficiency across several languages, including Python, making it suitable for various programming tasks. By 2023, GitHub Copilot broadened its repertoire to include support for R programming language (see Figure 2).

⁶ [Key GitHub Statistics in 2024 \(Users, Employees, and Trends\)](#) by Kinsta

⁷ [The top programming languages | The State of the Octoverse](#) by GitHub

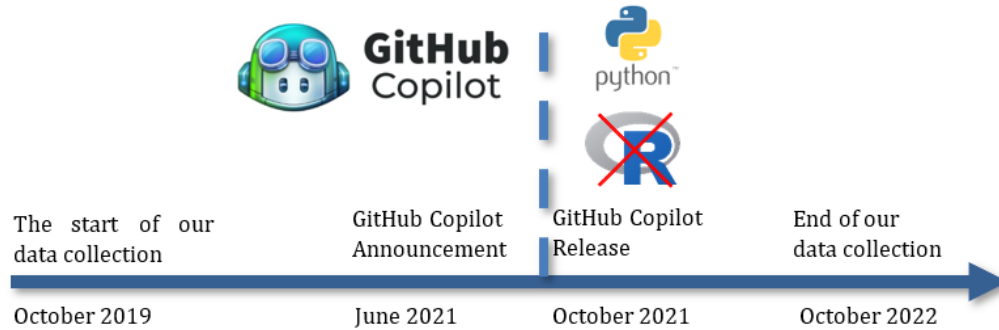


Figure 2. The illustration of the natural experiment

3.2 Data Collection

We compile two primary datasets on which the analyses are performed.

The Updates Dataset: We collected the version releases of packages and the timing of these releases, from October 2019 to October 2022, a period spanning two years before and one year after the introduction of GitHub Copilot. For Python packages, we focused our analysis on the 2000 most popular packages as of 2021 (van Kemenade et al. 2021)⁸. The detailed data on these 2000 packages were retrieved from the Python Package Index (PyPI), a central repository to which developers upload their open-source Python packages. Similarly, we collected data for all R packages from the Comprehensive R Archive Network (CRAN), a central repository for R packages⁹. We excluded packages that had not been updated in the two years preceding GitHub Copilot’s release, as they were unlikely to be influenced by this new tool. This resulted in the removal of 348 Python packages and over 7,965 R packages. We dropped the top 1% of the Python packages and the equal count of R packages, as they had extreme high activity pre-policy. After the matching (described in Section 3.1), we are left with 1610 Python and its matched R packages. We retrieved the package update data for the Python and R packages from PyPI and CRAN repositories, respectively, and aggregated the data at a quarterly level to account for the cyclic nature of package releases.

⁸ van Kemenade et al. (2021) aggregate and rank packages monthly on the [top-pypi-packages](#) repository.

⁹ Unlike Python, we did not find a reliable index of actively maintained R packages. Hence, we collected the data for all 21,000 R packages and dropped those that are either not actively maintained or are not open-sourced on GitHub.

The Commits Dataset: We used GitHub’s official API to programmatically download the commits data from the relevant GitHub repositories associated with the Python packages and R packages in our sample. As noted earlier, when developers “commit” a code change, they save the current state of their project, accompanied by a comment message detailing the changes. This process allows for collaboration and transparency in the development process. We kept in our sample only GitHub repositories that are associated with a single package resulted in 1089 Python packages and 1077 R packages.

Table 1(a) provides the variable description, and Table 1(b) provides the summary statistics.

Table 1(a). Variable Explanation

Variable	Description
Dependent Variables	
$VersionRelease_{it}$	The number of version releases of the package i in the quarter t
$Commits_{it}$	The number of new commits of the package i in the quarter t
$Commits_{ijt}$	The number of new commits of type j (maintenance or code development) of the package i in the quarter t
DiD Variables	
$PythonPackage_i$	A binary variable that carries the value of 1 if package i is a Python package
$afterCopilot_t$	A binary variable that carries the value of 1 if quarter t occurs after October 2021, i.e., after GitHub Copilot’s launch

Table 1(b). Summary Statistics

1610 packages in main analyses					
Variables	# Obs.	Mean	Std. Dev.	Min	Max
$VersionRelease_{it}$	38,279	1.12	2.42	0	192
$Commits_{it}$	26,428	23.54	67.69	0	3349
$Commits(j=CodeDevelopment)_{ijt}$	26,428	3.67	11.69	0	277
$Commits(j=Maintenance)_{ijt}$	26,428	11.2	38.9	0	3070

Note: Only 1089 python and 1077 R packages have the commit data in our dataset.

3.3 Commit Classification using Large Language Models

Annotating over 622,000 GitHub commits into specific categories is a challenging task given the absence of pre-defined criteria and established categorization for commits. GitHub commit messages are often unstructured and do not explicitly convey the type of change introduced. Understanding the types of code committed by developers is, however, central to understanding whether and how LLMs impact the trajectory of innovation on open-source projects. To address this challenge, we employed an LLM approach for efficient and accurate multi-label classification of commit comments.

Category Identification and Refinement: We randomly selected 500 Python and 500 R commit comments and asked GPT-4 to assign categories without providing explicit criteria and asked the model to justify the choice of category. This resulted in approximately 250 categories, many of which were similar in nature. To put these categories into broader umbrella buckets of related categories, we relied on the language capabilities of GPT-4 and the discernment abilities of a human expert. Specifically, we asked the expert to aggregate the categories based on their interpretation of the reasoning the LLM provided along with their independent judgment based on the commit comments and their research on the internet. This process provided us with 5 exclusive and comprehensive categories: “Maintenance” (including bug fixes, code cleanup, library dependency update, deprecation, refactoring etc.), “Code Development” (including feature addition, code optimization, etc.), “Documentation and Style” (including documentation, style update, etc.), “Testing and Quality Assurance” (including testing, version update, etc.) and “Other” (including license and copyright updates). For instance, “this commit fixes a bug in the error handling module” got tagged as *Maintenance*, “added a new feature for data visualization” as *Code Development*, “updating the README file with usage instructions” as *Documentation and Style*, “unit tests for the fixes introduced in #c1287” as *Testing and Quality Assurance*, and “updating the license file to the latest version” as *Other*.

Annotation Guidelines and Validation: Based on the recommendation by Lee et al. (2023), we developed detailed annotation guidelines to categorize the commits into five categories. Appendix Table B1 presents the annotation guidelines that were used with LLMs.

LLM Benchmarking and Selection: To choose the most suitable LLM model for our task, we benchmarked the performances of three different LLMs (GPT 4, GPT 3.5 Turbo, LLaMa 3 70B) against human annotations. We first randomly selected 200 Python commit comments and 200 R commit comments, which were tagged by two human annotators with an 87% agreement rate. In cases of conflicting tags, a third human annotator provided a tie-breaking vote. We then compared the outcome with human tags. Figure 3 shows the benchmarking results. As seen in Figure 3, GPT4 and GPT3.5 Turbo outperformed the open-source LLM and were very close to each other. For cost, time and efficiency reasons, we used

GPT 3.5 Turbo for our annotation task. Appendix Table B2 presents the model parameter details. By employing this systematic LLM-based annotation methodology, we were able to efficiently classify over 622,000 commit comments.

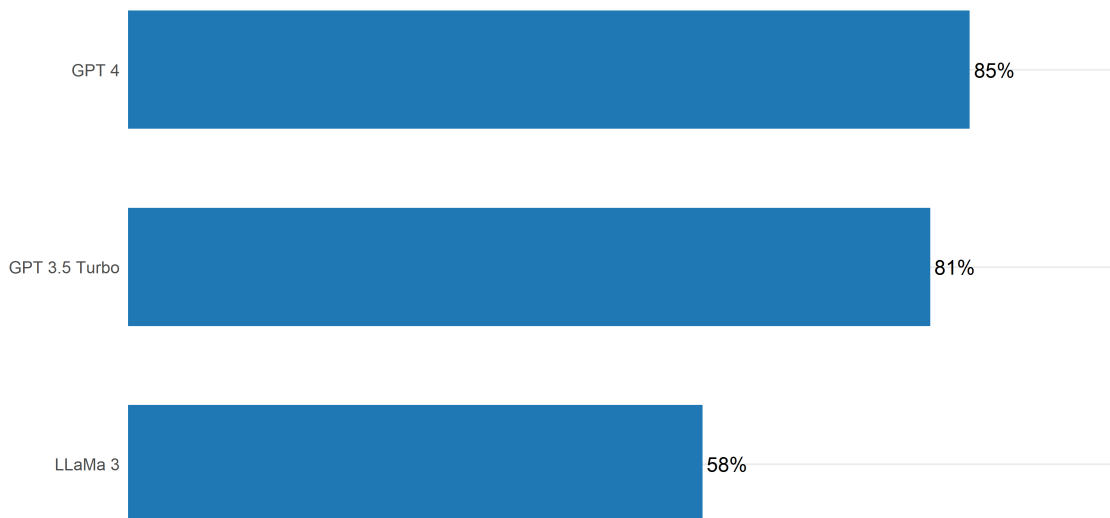


Figure 3. The accuracy of LLMs compared to human annotated ground truth.

4. Identification Strategies

4.1 Classical Propensity Score Matching with Difference in Differences Technique

Given that our natural experiment has a well-defined, treated group (i.e., Python packages) and a control group (i.e., R packages) with one-shot policy adoption, we employ the classic Difference-in-Differences (DiD) framework (Meyer 1995) and estimate the Average Treatment Effect on Treated (ATT) using a two-way fixed effects (TWFE) model. One potential threat to causal estimation using DiD in a natural experiment setting is that the assignment of treatment could be non-random. In our context, while the assignment of treatment (i.e., language supported by GitHub Copilot) was a business decision and is therefore exogenous to the package themselves, the choice for the package to be on a programming language is likely endogenous. To address the potential selection, we employed the *Propensity Score Matching* to pre-process the two package groups (Dehejia and Wahba 2002), a well-recognized technique in the literature. Specifically, we use the nearest neighbor method without replacement to obtain one control

package for every treated package (e.g., Mayya and Viswanathan 2024). We matched packages on the quarterly average of new version releases for each package prior to GitHub Copilot’s release, a proxy for GitHub activities. In Appendix C, we conduct a supplementary extension to this parametric matching technique by incorporating an additional constraint of exact matching on the package *categories* and find consistent results. Our final data set comprises 1610 unique Python packages, which serves as our treatment group, and 1610 unique R packages, which serves as our control group.

To study the changes in the quantity of innovation in packages that belong to the treatment group compared to those that belong to the control group, we set the dependent variable to be the count of new versions for package i in quarter t . We formally set up the model as follows:

$$\log (y_{it} + 1) = \alpha_i + \gamma_t + \beta \text{PythonPackage}_i X \text{AfterCopilot}_t + \eta_{it} \quad (1)$$

where y_{it} is the dependent variable of interest (such as the number of new versions) for package i in quarter t . In the model, α_i is the package-specific fixed effects; γ_t is the time-period fixed effects; PythonPackage_i carries the value of 1 if the package i is a Python package and AfterAdoption_t denotes the time after the launch of GitHub Copilot ($t \geq 0$). We cluster the standard errors by package in the TWFE analyses. The coefficient of interest is β , which estimates the difference between the Python and R packages after the launch of GitHub Copilot. Since we log-transform the dependent variable, β estimates the percentage change in the DV, consistent with all log-linear models.

4.1.1 Testing the Parallel Trends

To ensure that the treated and control packages have a similar trend before the release of GitHub Copilot, we employ the lead-lag model proposed by Autor (2003). Specifically, we treated the quarter before the launch of GitHub Copilot as the baseline and estimated the lead-lag model as follows:

$$\log (y_{it} + 1) = \alpha_i + \gamma_t + \sum \alpha_j^p \text{PythonPackage}_i X \text{RelativeQuarter}_j + \eta_{it} \quad (2)$$

where $RelativeQuarter_j$ is a vector of dummies that indicates Quarter j before and after the treatment took place. As per Autor (2003), the parallel trends assumption holds if all the coefficients (α_j^p) are not statistically significant for relative quarters j prior to the treatment. Figure 4 visualizes the outcomes of the model and demonstrates that the parallel trends assumptions are not violated.

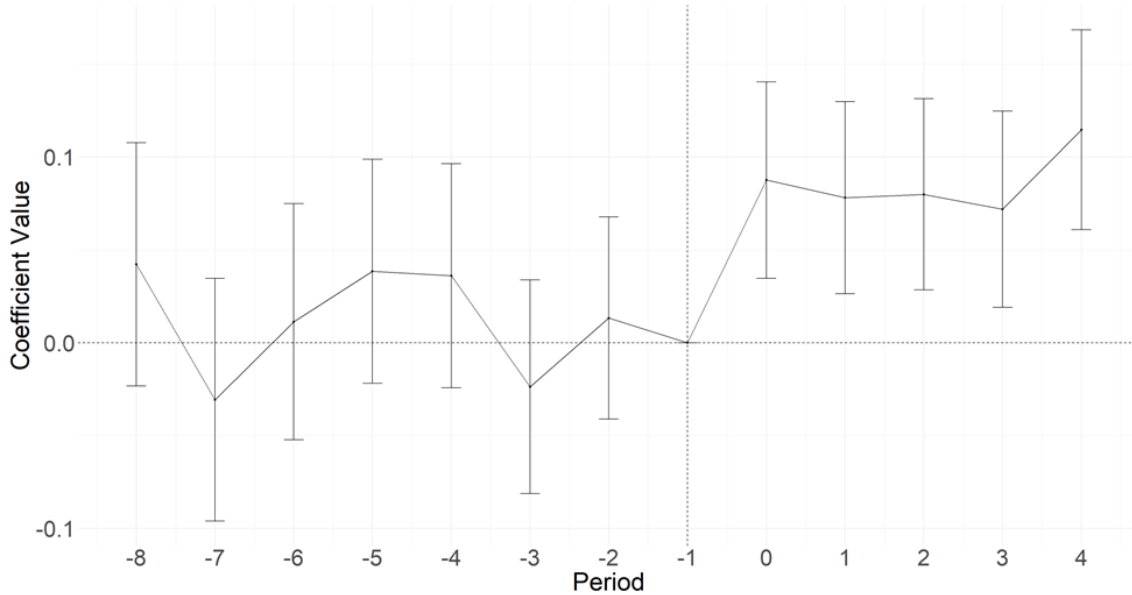


Figure 4. Parallel Trends Graph. (Confidence Interval at 99%)
 Note: The y-axis is the log transformation of the new version released each quarter

3.2 Synthetic Difference in Differences (SDiD) as an Alternative Identification Strategy

We extend our empirical analysis by employing Synthetic Difference-in-Differences (SDiD), a recent and cutting-edge framework introduced by Arkhangelsky et al. (2021). SDiD is a cutting-edge non-parametric estimation technique that relies on creating a “synthetic” control group (Abadie et al. 2010). By juxtaposing the treated unit with this synthetically generated control group, SDiD estimates the Average Treatment Effect (ATE). SDiD also generates weights to time-periods such that observations closer to the treatment periods receive higher weights compares to observations which are a few quarters away. This approach allows us to rigorously assess the impact of our intervention and has been applied in several recent studies (Berman and Israeli, 2022; Lambrecht et al., 2023).

Formally, we follow Arkhangelsky et al. (2021) and analyze a balanced panel with N units and T time periods, where the outcome for unit i in period t is denoted by y_{it} , and exposure to the binary treatment, the

release of GitHub Copilot, is denoted by $W_{it} \in \{0,1\}$. Moreover, assume that the first N_{co} (control) units (the R packages, in our case) are never exposed to the treatment, while the last $N_{tr} = N - N_{co}$ (treated) units (the Python packages) are exposed after time T_{pre} . As with synthetic controls (SC) methods, the model starts by finding weights SDiD that align pre-exposure trends in the outcomes of unexposed units with those of the exposed units. The model also looks for time weights $tsdid$ that balance the pre-exposure time periods with post-exposure ones. Then, the model uses these weights in a basic two-way fixed-effects regression to estimate the average causal effect. Formally, the SDiD estimation procedure solves:

$$(\hat{\tau}, \hat{\mu}, \hat{\alpha}, \hat{\beta}) = \underset{\tau}{\operatorname{argmin}} \left\{ \sum_{i=1}^N \sum_{T=1}^t (\log(y_{it} + 1) - \mu - \alpha_i - \beta_t - \text{AfterPython}_{it}\tau)^2 \hat{\omega}_i, \hat{\lambda}_t \right\} \quad (2)$$

where τ is the average ATT. The dependent variable, y_{it} , is the number of new versions that were released for package i in a specific quarter relative to the treatment period, t . Here, the model employs both the package-fixed effects and the period-fixed effects and AfterPython_{it} denotes Python package i in the periods after the launch of GitHub Copilot $t \geq 0$, as in the classic DiD framework. Equation (2) estimates a TWFE DiD model identical to Equation (1) with the addition of unit-specific weights i and time-specific weights t . The unit weights i are selected such that the weighted pre-treatment control outcomes have a similar trend to that of the average outcomes of the treatment units:

$$\hat{\omega}_0 + \sum_{i=1}^{N_{co}} \hat{\omega}_i \log \log(y_{it} + 1) \approx \frac{\sum_{i=N_{co}+1}^N \log \log(y_{it} + 1)}{|N_{tr}|}$$

The time weights t are constructed by making the pre-treatment time periods similar to the treatment periods for the control group:

$$\hat{\lambda}_0 + \sum_{t=-8}^{t=-1} \hat{\lambda}_t \log \log(y_{it} + 1) \approx \frac{\sum_{i=N_{co}+1}^N \log \log(y_{it} + 1)}{T_{post}}$$

To estimate the standard errors, we use the Jackknife Variance Estimation detailed in Arkhangelsky et al. (2021) and implemented in the R package named ‘‘SynthDiD.’’

5. Results

5.1 The Effect of LLMs on the Volume of Open-Source Innovation

For the first research question, which measures the impact of GitHub Copilot on the volume of innovation, we use the log-transformed value of count of updates per quarter as the dependent variable and estimate equation (1). Table 2 presents the estimation results of the ATT using the classical TWFE DiD (Column 1), a more restrictive TWFE DiD using the balanced panel sample because of the restriction of the SDiD strategy (Column 2), and the SDiD estimation (Column 3). The coefficient of 0.164 in column (1) means that GitHub Copilot increases the number of new package releases by 17.82% (calculated as $(e^{0.164} - 1) * 100 = 17.82\%$). The results in columns (2) and (3) are consistent: the coefficients reflect a 9.52%, and 8.54% increase in the number of new Python package versions after the release of GitHub Copilot, compared to their matched R counterpart.

Table 2. The Effect on the Volume of Innovation

Dependent Variable	Number of New Versions			Number of New Commits		
	(1) <i>TWFE</i>	(2) <i>Balanced TWFE</i>	(3) <i>SynthDiD</i>	(4) <i>TWFE</i>	(5) <i>Balanced TWFE</i>	(6) <i>SynthDiD</i>
<i>PythonPackage_i X afterCopilot_t</i>	0.164*** (0.012)	0.091*** (0.012)	0.082*** (0.011)	0.417*** (0.035)	0.314*** (0.037)	0.288*** (0.0351)
Time Fixed Effect	YES	YES	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES	YES	YES
# of Obs.	38,279	32,045	32,045	26,428	22,451	22,451

Note: *** p<0.001, ** p<0.01, * p<0.05, + p<0.1

Standard Errors clustered around package in parentheses for (1), (2), (4) and (5). The variance for (3) and (6) are estimated using the Jackknife variance procedure.

The observations vary between models (1) and (2) because of the unbalanced nature of the data. The observations vary across model (1)-(3) and (4)-(6) because only 1089 python and 1077 R packages out of 1610 packages have the commits data on GitHub, and which can be associated with just that package.

As an alternative measurement of the volume of innovation, we use the number of new commits received each quarter. Columns (4) through (6) of Table 2 present the results with log-transformed value of commits as the DV. The results are consistent and numerically much larger than the number of new releases. For instance, column (4) suggests that the launch of GitHub Copilot increases the commits by 51% (calculated as $(e^{0.417} - 1) * 100 = 51\%$) for Python packages compared to its R counterpart. At a mean value of about 23.3 commits per quarter, this improvement translates to 11.8 additional commits per quarter. We find consistent results with the balanced TWFE and SDiD analyses in columns (5) and (6), respectively.

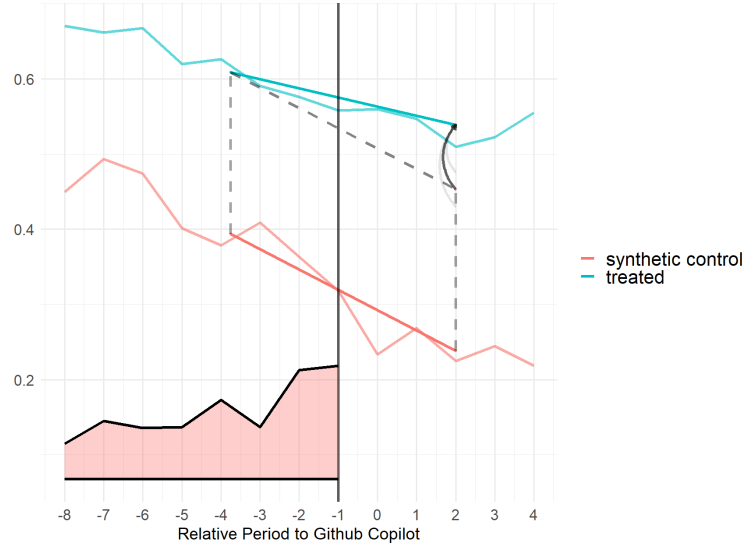


Figure 5a. The Impact of LLMs on the package release each quarter.

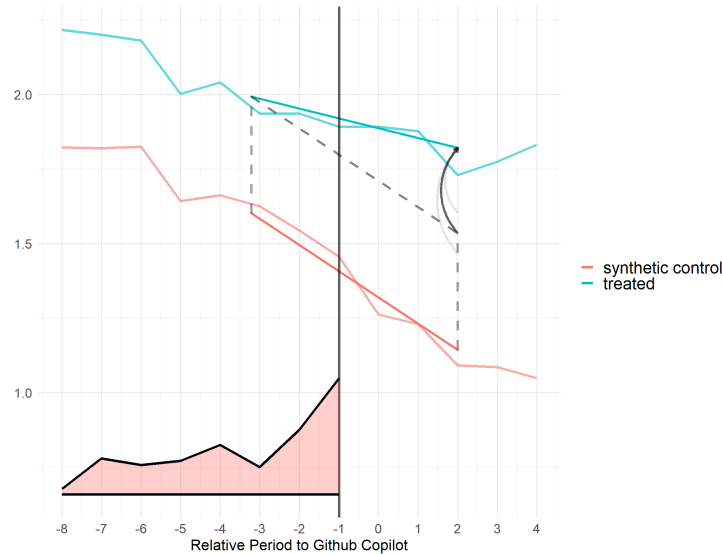


Figure 5b. The Impact of LLMs on the count of commits each quarter.

Note: The x-axis indicates the relative period to the launch of GitHub Copilot (October 2021)

Figure 5 visualizes the results of the synthetic DiD analysis. The vertical axes quantify the logarithmic transformation of the count of new version releases and count of new commits respectively, which enable a multiplicative interpretation of changes over time. The horizontal axis marks the period relative to the introduction of GitHub Copilot, with the point “0” designating its launch. The turquoise line represents the logarithmic trend of the treatment effect for the treated group (Python Packages) in the post-treatment period. The shaded red area delineates the trend for the synthetic control group (R Packages), which is a weighted combination of R packages used to construct the counterfactual scenario for Python packages (the

time weights are presented in the shaded red area at the bottom of the graph). The dashed black line indicates the inferred counterfactual for the treated group, had GitHub Copilot not been introduced, allowing for a direct comparison of the actual and the estimated absence of the treatment. The divergence between the blue line and the dashed black line post-introduction reflects an approximate 8.54% increase in the number of new versions and 33.37% increase in the count of commits in Python relative to the control group.

5.2 The Effect of LLMs on the Type of Innovation

To study how the programming focused LLMs affect two of the most relevant programming tasks (code development and maintenance), we modify equation (1) into a triple difference model to contrast the trajectory of maintenance commits with code development commits. We formalize the model as follows:

$$\begin{aligned} \log(y_{ijt} + 1) = & \alpha_{ij} + \gamma_t + \beta_1 PythonPackage_i X afterCopilot_t \\ & + \beta_2 PythonPackage_i X CategoryMaintenance_j X afterCopilot_t \quad (3) \\ & + \gamma_t X CategoryMaintenance_j X afterCopilot_t + \eta_{ijt} \end{aligned}$$

where i indexes the package, j indexes the category and t indexes the quarter. The dependent variable, y_{ijt} is the number of commits for the package i made in the category j and in a quarter t . We introduce a triple difference term (i.e., $PythonPackage_i X afterCopilot_{it} X Category_Maintenance_j$) where the dummy $Category_Maintenance_j$ carries a value of 1 for the maintenance commit count and 0 for the code development commit count.

We present the results in Table 3. Our TWFE estimation reveals that, while code development commits increase by 16.76% (calculated as $(e^{(0.155)} - 1) * 100 = 16.76\%$), the increase in maintenance commit surpasses the features development commits by 15.14% (calculated as $(e^{(0.141)} - 1) * 100 = 15.14\%$)¹⁰. We find similar results using Balanced TWFE model, as shown in column (2) of the table.¹¹ Overall, we see a persistent difference in the increase in maintenance compared with the increase in code development. Finally, we find

¹⁰ The maintenance contributions are significantly higher compared to the increase in code development contributions. The test for significance for the difference between the coefficients yields t statistic of 3.77.

¹¹ The Synthetic Difference in Difference model does not accommodate a Triple Difference Specification.

similar results while estimating SDiD on the sub-samples, as shown in Appendix Figure A1. We replicate equation (1) by running two independent analyses: one each with the count of maintenance commits and the count of code development commits as the dependent variables. We present the results of the sub-sample analysis in Appendix Table A1 and visualize the outcomes in a graph, as shown in Figure 6. The analysis further provides assurance that maintenance related commits see a massive increase, over and above the increase witnessed in code development related commits.

Table 3. The Effect on the Type of Innovation

Dependent Variable	Number of New Commits	
Models	(1)	(2)
	<i>TWFE</i>	<i>Balanced TWFE</i>
<i>PythonPackage_i X afterCopilot_t</i>	0.155*** (0.023)	0.101*** (0.024)
<i>PythonPackage_i X afterCopilot_t</i> <i>X Category Maintenance_i</i>	0.141*** (0.018)	0.126*** (0.020)
Time Fixed Effect	YES	YES
Package Fixed Effect	YES	YES
# of Obs.	52,856	44,902

Note: Standard Errors clustered around package in parentheses
 *** p<0.001, ** p<0.01, * p<0.05, + p<0.1

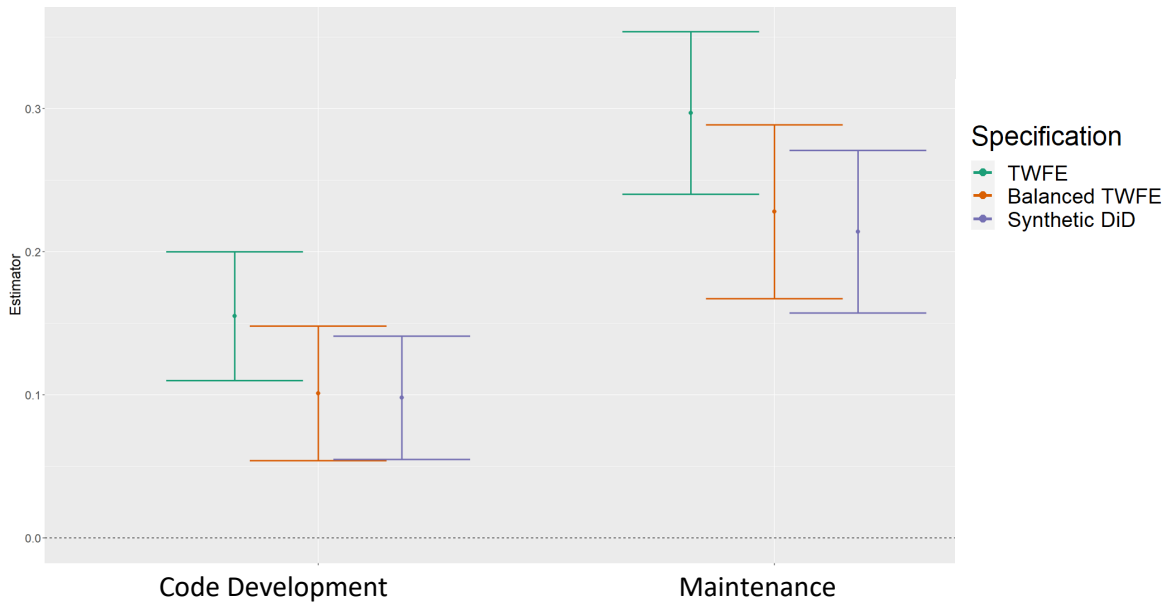


Figure 6. The Impact of LLMs on the Type of Innovation

Note: The y-axis is the logged value of the count of the commits for each category in each quarter.

5.3 A deep dive into LLM-Augmented Contributions with better Context Provision

As noted earlier, contributors may be drawn to use LLMs on projects with more user activity footprint, given the abundant resources available to facilitate the creation of precise prompts for LLMs. Alternatively, simpler projects with fewer dependencies and clearer features may better capitalize on LLM assistance. These projects often benefit from a more straightforward review process and quicker turnaround, making them more conducive to experimentation with LLM-generated solutions. A priori, it is unclear whether LLMs affect projects with above or below-median user activities.

To understand how these two types of contributions vary depending on user activity footprint,¹² we broke down the analysis into sub-samples based on the project-activity levels (proxied by the pre-treatment package release volume). After splitting the packages into two groups based on the median volume of pre-treatment activity, we use equation (3) that utilizes the triple difference model to contrast the trajectory of maintenance commits compared to code development commits for the above or below median sub-sample.

We present the results in Table 4. We observed a consistent pattern: in both the above-median and below-median user-activity packages, *maintenance* contributions significantly outweigh *code development* contributions. However, the gap between these two types of contributions is starker for packages with above-median user activities. Specifically, contrasting the DiD coefficients in columns (1) and (3), the difference between the code development and maintenance contributions is starker for the above-median projects (column 1) compared to the below-median projects (column 3). Specifically, while more-active projects saw a 35.93% increase in *code development* compared to less-active projects (at 2.83% statistically insignificant increase), they saw a significantly larger increase in *maintenance* commits at 22.01%, compared to the increase in less-active projects (at 8.32%).

Together, these results suggest that Copilot’s assistance is the most beneficial for maintenance and refinement activities in projects with high coding related activity, which tend to provide rich context to

¹² Our primary measure of contribution activity footprint is the average pre-treatment release count. We also use average pre-treatment commit count as an alternative measure and find consistent results (unreported)

LLMs. The shift in concentration towards projects with high activity underscores the symbiotic relationship between AI capabilities and the availability of rich, human-generated context. Popular packages are more likely to attract feature requests and bug reports in their discussion forums. These exchanges often result in a detailed discussion where both contributors and users provide feedback before the requests are formalized. Furthermore, these packages often have detailed documentation and a clear and modular codebase, enabling contributors to get a much-detailed context before starting the contribution. All these points justify a larger jump in both *code development* and *maintenance* commits. Finally, as noted above, *maintenance* tasks, by their very nature, are characterized by well-defined solution-spaces. This inherent characteristic leads to a much larger boost in *maintenance* commits compared to *code development*, even in the presence of thorough discussions and detailed documentation. Our findings also highlight the risk that LLM introduction might amplify existing knowledge disparities in open-source communities.

Table 4. The Effect on the type of Innovation—Subsample based on Popularity

Dependent Variable	Number of New Commits			
	Above Median Package		Below Median Package	
Models	(1) TWFE	(2) <i>Balanced</i> TWFE	(3) TWFE	(4) <i>Balanced</i> TWFE
$PythonPackage_i X afterCopilot_t$	0.307*** (0.039)	0.248*** (0.042)	0.028 (0.017)	0.022 (0.019)
$PythonPackage_i X afterCopilot_t$ $X Category Maintenance_i$	0.199*** (0.031)	0.164*** (0.035)	0.08*** (0.020)	0.095*** (0.022)
Time Fixed Effect	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES
# of Obs.	26072	21,424	26,784	23,478

Note: Standard Errors clustered around package in parentheses
 *** p<0.001, ** p<0.01, * p<0.05, + p<0.1

5.4 Robustness Checks

We perform several additional analyses, apart from Synthetic DiD. We repeated our analyses by estimating using alternative model specifications (non-logged dependent variables), testing for temporal variation (dropping initial quarters, and shifting treatment timing) and ruling out some alternative explanations including the commit rollbacks.

5.4.1 Alternative Model Specification: Raw value of Dependent Variable

Appendix Table A2 presents our analysis for the impact of LLMs on the volume of open-source innovation using the raw value of the dependent variable instead of the logged version. The TWFE DiD estimation (column 1) shows a 0.248 boost in quarterly new version releases for Python packages compared to matched R packages. This effect is robust across specifications, with the balanced panel TWFE and SDiD models showing increases of 0.11 and 0.11, respectively (columns (2) and (3)). With the count of commits as the dependent variable, the TWFE DiD model indicates a 6.296 increase in commits for Python packages (column (4)). The increase is consistent across balanced TWFE and SDiD analyses in columns (5) and (6), respectively. Similarly, Appendix Tables A3 and A4 replicates Tables 3 and 4 but with the raw value of the dependent variables. The results are consistent with the main tables, giving us confidence that the results are robust to alternative model specifications.

5.4.2 Temporal Variations: Dropping Initial three Quarters (0,1 and 2)

To avoid selection from early adopters of GitHub Copilot that may result in unique usage patterns, we perform the analysis by dropping the initial 3 quarters when GitHub Copilot was in its technical preview period. We present the results of the first specification in Appendix Table A5. The TWFE DiD estimation (column 1) shows a 21.4% boost in quarterly new version releases (calculated as $(e^{0.194} - 1) * 100 = 21.4\%$) for Python packages compared to matched R packages. This effect is robust across specifications, with the balanced panel TWFE and SDiD models showing increases of 10.1% and 9%, respectively (columns (2) and (3)) as well as with the count of commits as the dependent variable. Similarly, Appendix Tables A6 and A7 replicates Tables 3 and 4 but after dropping the quarters 0 through 2, as in Appendix Table A5. The results are consistent with the main specification and have larger coefficients, giving us confidence that the results are not primarily driven by the initial adopters of GitHub Copilot. The larger coefficients, in fact, point towards an accelerating trend of contributions towards Python packages owing to LLM assistance.

5.4.3 Temporal Variations: Moving the Treatment to June 2021

While GitHub Copilot was made available via various IDEs in October 2021, the announcement happened in June 2021 with a selective set of programmers receiving early access. Hence, we advance our treatment window by one quarter (to June 2021) to test the robustness of our estimation. Appendix Table A8 presents our analysis of the impact of LLMs on the volume of open-source innovation. The TWFE DiD estimation (column 1) shows a 17.82% boost in quarterly new version releases (calculated as $(e^{0.164} - 1) * 100 = 17.82\%$) for Python packages compared to matched R packages. This effect is robust across specifications, with the balanced panel TWFE and SDiD models showing increases of 9.52% and 9%, respectively (columns (2) and (3)), as well as across the count of commits as the DV. Similarly, Appendix Tables A9 and A10 replicate Tables 3 and 4, but with June 2021 as the treatment quarter. The results are consistent with the main specification.

5.4.4 (Ruling out) Alternative Explanation: Analysis using Commit Rollbacks

Finally, to rule out the possibility that the increase in maintenance contributions is driven from an increase in the number of errors introduced by GitHub Copilot itself, we explore the effect of the release of GitHub Copilot on the number of rollbacks. Specifically, we conduct a difference-in-differences analysis where the dependent variable is the number of commits with commit comments containing the word “rollback,” referring to reversing a previously committed change to the GitHub repository. The estimation result for this analysis is presented in Appendix Table A11. We find that the number of rollbacks did not significantly change after the release of GitHub Copilot.

6. Discussion and Conclusion

6.1 Discussion

Our work provides critical new insights regarding the causal effects of LLMs on innovation and provides a much-needed framework for understanding *what to look for in a task to predict the impact of GenAI on that task*, a question marked with contrasting outcomes in literature. As LLM adoption accelerates among programmers and software developers, understanding the impact of such LLMs on innovation in this space is critical. Extant research has highlighted the positive impact of LLMs on the swiftness of programming

in a guided environment, such as specific tasks in organizational settings (e.g., Peng et al. 2023). However, it is imperative to critically analyze whether such LLMs can impact the evolving dynamics in collaborative space such as open-source, which extends beyond the ease of programming and is driven by self-selected, high-skilled volunteers to push the boundaries of existing knowledge. Furthermore, open-source innovation tasks are unguided and are much more rapid compared to organizational closed-source innovation (Paulson et al. 2004), needing a healthy mixture of tasks needing *out-of-the-box* extrapolative solutions and *within-the-box* interpolative solutions. Given that LLMs have been shown to affect knowledge exchange on community-driven platforms like *Stack Overflow* (e.g., Burtch et al. 2024, Quinn and Gutt 2023), it is critical to investigate whether LLMs effect community-driven innovation setting such as open-source.

To our knowledge, ours is the first to leverage the natural experiment of GitHub Copilot’s staggered language support for this purpose. By leveraging a unique natural experiment—the limited launch of GitHub Copilot in October 2021 to support languages like Python, without supporting languages like R—we empirically identify the causal effects of LLMs on programmers’ contributions to open innovation: on the volume of innovation, the type of innovation and on the distribution of innovation across various projects. We employ a variety of econometric techniques, including the classical Two-Way Fixed Effects and Synthetic Difference in Differences techniques, to causally identify the impact.

We found that the availability of LLM support increased contributions in an open-source environment. Not surprisingly, this increase in the new update releases is driven by a large increase in the number of new commits that contributors make to the package. This finding augments burgeoning research on LLMs by uncovering two key aspects: first, LLMs not only add to the individual productivity enhancement among guided programming tasks (e.g., Peng et al. 2023), but also enhance collaborative open-innovation tasks, where contributions are voluntary and unguided. Second, LLMs not only help users that are less experienced or novices (e.g., Brynjolfsson et al. 2023, Noy and Zhang 2023) but also have a positive impact on expert programmers that contribute to collaborative projects. Our findings are encouraging to coders, software industry, collaborative communities, and policymakers, many of whom are concerned about the breakneck speed at which LLMs impact various aspects of productivity and community.

Next, our investigation about the type of innovation reveals that LLMs improve both iterative contributions via *maintenance* as well as origination contributions via *code development*. However, LLMs disproportionately enhanced maintenance contributions (e.g., debugging) over code development contributions (e.g., algorithm implementation). The finding can be explained by understanding the benefits that LLMs bring in a problem-solving process. LLMs are fundamentally trained to predict the next “token” by comprehending the context of the preceding tokens leading up to the token to be predicted. Given the contextual understanding of any LLM to solve the problem at hand, an LLM can offer precise guidance related to the problem areas which are inherently suited to generate a detailed context. Maintenance solutions, by definition, have a much-detailed context because of a well-defined problem definition. For instance, GitHub’s bug reporting mechanism requires bug reporters to provide a thorough explanation of the issue, along with version details and instructions for reproducing errors. This detailed report and error reproduction can further enhance the prompt generation for a programming focused LLMs, refining their contextual understanding and enabling a more precise solution. Essentially, GitHub’s problem reporting mechanism delineates a well-defined solution space in which LLMs prove highly effective.

These findings provide robust empirical support to the intuition that LLMs excel in iterative tasks needing interpolative solutions, where consulting past solutions helps solve current task. LLM support in such tasks may be akin to having an “oracle” to indicate solution correctness, thus lessening cognitive burdens without introducing new errors. In contrast, LLMs may have a modest impact on origination tasks needing extrapolative solutions, such as the design of complex systems, which require higher-level abstract thinking, putting them outside the well-defined solution spaces that LLMs handle well. These findings further align with prior observations regarding LLMs’ conflicting effects on less-structured domains (e.g., Chen and Chan 2023, Zhou and Lee 2024).

Finally, our deep dive into the heterogeneity based on user activities revealed that the disparity between maintenance and code development activities was particularly pronounced in more-active packages, which contain more discussions and documentation (i.e., richer context). It highlights a disparity between communities with an above-median and below-median user activity footprint. While the gap between code

development commits is already larger for the above median packages, the maintenance related updates further widen the gap. These findings reinforce our argument about the ease with which detailed contextual information is generated. The increased level of bug reports and discussions on popular packages, the detailed documentation and the widespread usage by end users explains this gap. More popular packages cater to a broader end-user base, which increases the likelihood of exposing corner cases in certain programming logic, resulting in crashes or undesirable outcomes. Naturally, such undesirable outcomes get reported on GitHub repositories, making them more visible to contributors.

Our findings have significant implications for the trajectory of innovation, as LLM capabilities continue to evolve. The concentration of LLM-driven contributions in active projects with more discussions and documentation, which offer richer context for LLMs to provide solutions, suggests that the gap between *iterative* and *origination* tasks may persist or even widen as LLMs improve their ability to process larger context windows and are better trained to understand the context. This potential divergence in the rate of improvement between *iterative* and *origination* tasks could reshape the landscape of innovation, with LLMs playing an increasingly dominant role in tasks that can be solved within existing knowledge frameworks. Such tasks might include, for example, understanding customer concerns and identifying solutions on the basis of prior similar cases, verifying the accuracy of accounting data, or retrieving semi-related documents among millions of legal documents. Our findings also add to the burgeoning literature examining the impact of AI on user behavior (e.g., Sun et al. 2024), highlighting the nuanced ways in which AI capabilities can shape human decision making.

6.2 Implications

Our project has many theoretical and practical implications. To the best of our knowledge, we are among the first to leverage the natural experiment related to the rollout of GitHub Copilot to answer questions related to the impact of LLMs on open innovation. Unlike some of the earlier findings that report a precipitous drop in knowledge dissemination and exchange on User Generated Platforms (UGCs), we note an increase in the open-source commits after the LLM release. Furthermore, we extend extant research on

programming productivity by noting that productivity increases do not just benefit private enterprises with guided tasks but also unguided open-source innovation. These results are encouraging and provide guidance to other open-source platforms. The results motivate other open-source platforms to provide the community with the platform-specific LLM to encourage more contributions from the community.

Another key implication concerns the extent to which LLMs are effective in software development tasks. Most research, including Peng et al. (2023), notes that LLMs enhance an individual's ability to complete guided tasks. These experimental programming tasks can be viewed as *origination* tasks, where developers generate a solution from scratch and are expected to complete the task on their own. However, our findings indicate that LLMs are more effective for *iterative* tasks, where understanding others' work from the prior iteration is necessary before making their contribution in the current iteration. This raises an interesting possibility: the overall impact of LLMs on collaborative software development, such as in open-source projects, could be stronger than initial experimental results suggest.

The question that arises is whether, in light of the rising popularity of LLMs, workers using LLMs to generate solutions may be disproportionately drawn to tasks with readily verifiable answers and detailed backgrounds. Such a tendency may lead to a downward trajectory in high-impact solutions. In other words, the ease and efficiency of addressing well-defined problems could overshadow the pursuit of groundbreaking discoveries. Accordingly, our results highlight the need for policies and strategies that promote extrapolative thinking in the innovation domain. Such strategies might include incentives, such as grants, awards, or recognition programs. This approach could create a balance between human ingenuity and LLM capabilities, utilizing the strengths of each to drive a varied and impactful innovation scene, even as LLMs continue to advance. Such policy interventions could also influence the potential trade-offs between productivity gains and the displacement of human workers (Agrawal et al. 2023) and may foster a balance between technological advancement and human capital development.

Our findings are also important for firms that participate in or encourage their employees to contribute to community driven open-source projects. Given the LLMs' disproportionate impact on maintenance on more established projects, firms must devise strategies to incentivize senior and experienced contributors

to engage with early-stage projects needing *out-of-the-box* architectural thinking. This could include mentorship programs, grants, or incentivization for contributors.

6.3 Limitations

Our current findings have certain limitations common across all research related to innovation. Our commits only capture successful pull requests, in that the project’s owner accepted the changes some contributor made. The rejected pull requests, just as failed innovation efforts elsewhere, are not easily visible, which may bias the results. We are investigating ways to address this bias, which may also be an exciting avenue for future work. Another notable limitation of our analysis is that not every contributor might have used Copilot, meaning our findings likely represent a lower bound of the true effect. As LLM adoption increases, we expect an even more significant effect on innovation patterns. To this end, our results present a lower bound. Based on GitHub’s announcement, we know that the adoption is in millions. As a next step, researchers could work with GitHub to obtain information about the sign-up details on GitHub Copilot. Additionally, generalizing these findings to other innovation domains than open-source requires caution, as the unique characteristics of open-source innovation, such as voluntary contributions and high degree of autonomy given to contributors, may not fully translate to other settings with varied motivations and organizational structures.

Limitations notwithstanding, our current findings have significant implications for open innovation. The imminent step for this research agenda is understanding the mechanism by which LLMs disproportionately affect iterative contributions via maintenance over origination contributions via code development. It could be that GitHub copilot makes it easier for contributors by instilling confidence about identifying the flaw in the logic when working with an AI “pair programmer.” Extant research on pair programming suggests that such a practice reduces subtle errors that make debugging much more difficult (Cockburn and Williams 2000). The reduction in the cognitive load of finding and rectifying complex bugs may be driving these findings.

References

- Abadie A, Diamond A, Hainmueller J (2010) Synthetic control methods for comparative case studies: Estimating the effect of California's tobacco control program. *Journal of the American statistical Association*. 105(490):493-505.
- Agrawal A, Gans JS, Goldfarb A (2023) Do we want less automation? *Science*, 381(6654), pp.155-158.
- Arkhangelsky D, Athey S, Hirshberg DA, Imbens GW, Wager S (2021) Synthetic difference-in-differences. *American Economic Review*, 111(12), 4088-4118.
- Autor DH (2003) Outsourcing at will: The contribution of unjust dismissal doctrine to the growth of employment outsourcing. *Journal of labor economics*, 21(1), pp.1-42.
- Belenzon S, Schankerman MA (2008) Motivation and sorting in open-source software innovation.
- Berman R, Israeli A (2022) The value of descriptive analytics: Evidence from online retailers. *Marketing Science*, 41(6), 1074-1096.
- Brynjolfsson E, Mitchell T, Rock D (2018) What can machines learn and what does it mean for occupations and the economy?. In *AEA papers and proceedings* (Vol. 108, pp. 43-47). 2014 Broadway, Suite 305, Nashville, TN 37203: American Economic Association.
- Brynjolfsson E (2022) The Turing trap: The promise and peril of human-like artificial intelligence. <https://doi.org/10.48550/arXiv.2201.04200>
- Brynjolfsson E, Li D, Raymond LR (2023) Generative AI at work (No. w31161). *National Bureau of Economic Research*.
- Burtch G, Lee D, Chen Z (2024) The consequences of generative AI for online knowledge communities. *Scientific Reports* 14(1):10413.
- Chen Z, Chan J (2023) Large language model in creative work: The role of collaboration modality and user expertise. *Available at SSRN 4575598*.
- Cockburn A, Williams L (2000) The costs and benefits of pair programming. *Extreme programming examined*, 8, pp.223-247.
- Curtis B, Krasner H, Iscoe N (1988) A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268-87.
- Doshi AR, Hauser OP (2024) Generative artificial intelligence enhances individual creativity but reduces the collective diversity of novel content. *Science Advances* 10(28)
- Eloundou, T, Manning, S, Mishkin, P, & Rock, D (2024) GPTs are GPTs: Labor market impact potential of LLMs. *Science*, 384(6702), 1306-1308.
- Fershtman C, Gandal N (2011) Direct and indirect knowledge spillovers: The “social network” of open-source projects. *The RAND Journal of Economics*, 42(1), 70-91.
- GitHub (2023) The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot. *GitHub Blog*. <https://github.blog/2023-06-27-the-economic-impact-of-the-ai-powered-developer-lifecycle-and-lessons-from-github-copilot/>
- Horton JJ (2023) Large language models as simulated economic agents: What can we learn from homo silicus? (No. w31122). *National Bureau of Economic Research*.

- Kalliamvakou E (2022) Research: quantifying GitHub Copilot's impact on developer productivity and happiness. *The GitHub Blog*, Retrieved on 4th January 2024 from <https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>
- Lambrecht A, Tucker C, Zhang X (2023) TV advertising and online sales: A case study of intertemporal substitution effects for an online travel platform. *Journal of Marketing Research*. <https://doi.org/10.1177/00222437231180171>
- Larman C (2004) Agile and iterative development: a manager's guide. *Addison-Wesley Professional*
- Lee S, DeLucia A, Nangia N, Ganedi P, Guan R, Li R, Ngaw B, Singhal A, Vaidya S, Yuan Z, Zhang L Sedoc J (2023) Common law annotations: Investigating the stability of dialog system output annotations. In *Findings of the Association for Computational Linguistics: ACL 2023* (pp. 12315-12349).
- Lerner J, Tirole J (2002) Some simple economics of open source. *The Journal of Industrial Economics*, 50(2), 197-234.
- Maruping LM, Daniel SL, Cataldo M (2019) Developer centrality and the impact of value congruence and incongruence on commitment and code contribution activity in open source software communities. *MIS Quarterly*, 43(3), 951-976.
- Mayya R, Viswanathan S (2024) Delaying Informed Consent: An empirical investigation of mobile apps' upgrade decisions. *Management Science (Forthcoming)*.
- Medappa PK, Tunc MM, Li X (2023) Sponsorship funding in open-source software: Effort reallocation and spillover effects in knowledge-sharing ecosystems. *SSRN*. <https://ssrn.com/abstract=4484403>
- Meyer BD (1995) Natural and Quasi-experiments in Economics. *Journal of Business and Economic Statistics* 13(2): 151-161
- Noy S, Zhang W (2023) Experimental evidence on the productivity effects of generative artificial intelligence. *Science*, 381(6654), 187-192. doi:10.1126/science.adh2586
- Paulson JW, Succi G, Eberlein A (2004) An empirical study of open-source and closed-source software products. *IEEE transactions on software engineering*, 30(4), pp.246-256.
- Peng S, Kalliamvakou E, Cihon P, Demirel M (2023) The impact of AI on developer productivity: Evidence from github copilot. *arXiv*. <https://doi.org/10.48550/arXiv.2302.06590>
- Quinn M, Gutt D (2023) The effect of generative artificial intelligence on QandA platforms – Consequences for questions, answers, question readability, and novelty. *SLUB*. <https://katalog.slub-dresden.de/en/id/0-1859650120>
- Riveros I, Zhang M, Luo L (2023) How does generative AI affect demand for user-generated content? Evidence from Stack Exchange (June 28, 2023). *SSRN Electronic Journal*. <https://ssrn.com/abstract=4494646>
- Schockaert S, Prade H. (2013) Interpolative and extrapolative reasoning in propositional theories using qualitative knowledge about conceptual spaces. *Artificial Intelligence*, 202, pp.86-131.
- Sun C, Shi Z, Liu X, Ghose A (2024) The Effect of Voice AI on Digital Commerce. *Information Systems Research (Forthcoming)*
- Tan Z, Beigi A, Wang S, Guo R, Bhattacharjee A, Jiang B, Karami M, Li J, Cheng L, Liu H (2024) Large Language Models for Data Annotation: A Survey. *arXiv preprint arXiv:2402.13446*.
- van Kemenade H, Thoma M, Si R, Dollenstein Z (2021) hugovk/top-pypi-packages: Release 2021.10 (2010.10). Zenodo. <https://doi.org/10.5281/zenodo.10730812>

von Krogh G, Haefliger S, Spaeth S, Wallin MW (2012) Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, 36(2), 649-676.

Zhou E and Lee DK (2024). Generative Artificial Intelligence, human creativity, and art. *PNAS Nexus* 3(3), pp.052.

Appendix A

Appendix Figures

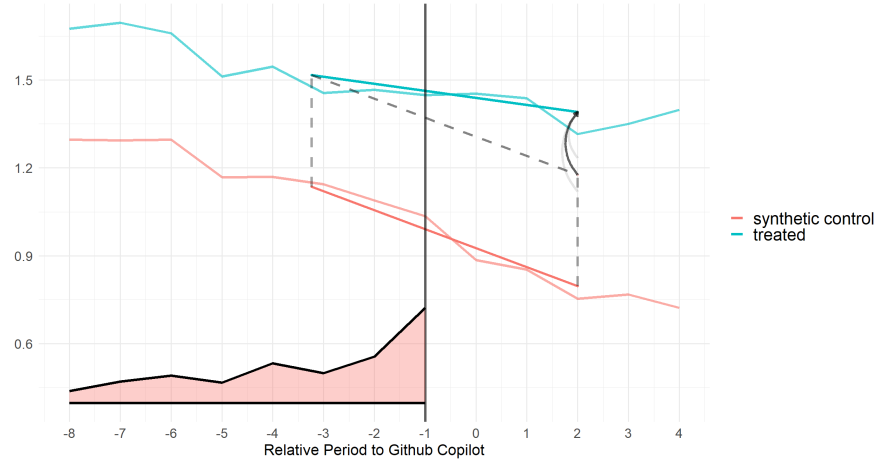


Figure A1a. The impact of Copilot on the count of *maintenance* commits in each quarter.

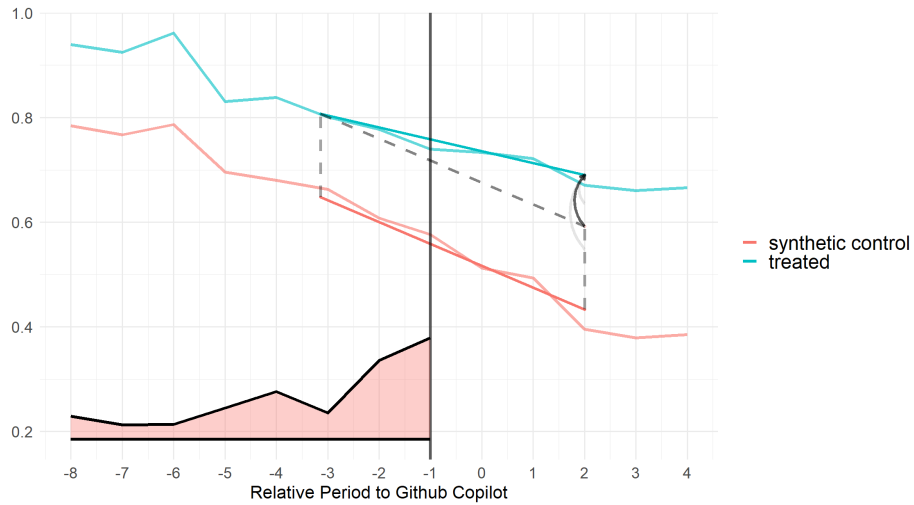


Figure A1b. The impact of Copilot on the count of code development commits in each quarter.

Note: The x-axis indicates the period relative to the launch of GitHub Copilot (October 2021)

Appendix Tables

Table A1. The Effect on the Type of Innovation – Sub Sample Analysis

Dependent Variable	Maintenance			Code Development		
	(1) <i>TWFE</i>	(2) <i>Balanced TWFE</i>	(3) <i>SynthDiD</i>	(4) <i>TWFE</i>	(5) <i>Balanced TWFE</i>	(6) <i>SynthDiD</i>
<i>PythonPackage_i X afterCopilot_{it}</i>	0.297*** (0.029)	0.228*** (0.031)	0.214*** (0.029)	0.155*** (0.023)	0.101*** (0.024)	0.098*** (0.022)
Time Fixed Effect	YES	YES	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES	YES	YES
# of Obs.	26,428	22,451	22,451	26,428	22,451	22,451

Note: Standard Errors clustered around package in parentheses for columns (1) and (2);
 *** p<0.001, ** p<0.01, * p<0.05

This table provides the values for Figure 6 in the main manuscript.

Table A2. The Effect of GitHub Copilot on the Volume of Innovation – Raw Dependent Variable

Dependent Variable	Number of New Version Releases			Number of New Commits		
	(1) <i>TWFE</i>	(2) <i>Balanced TWFE</i>	(3) <i>SDiD</i>	(4) <i>TWFE</i>	(5) <i>Balanced TWFE</i>	(6) <i>SDiD</i>
<i>PythonPackage_i X afterCopilot_{it}</i>	0.248*** (0.0534)	0.11* (0.05)	0.11* (0.049)	6.296*** (1.393)	4.107** (1.495)	4.02*** (1.280)
Time Fixed Effect	YES	YES	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES	YES	YES
# of Obs.	38,279	32,045	32,071	26,428	22,451	22,451

Note: This table contains presents a robustness check where the dependent variable is not logged (i.e., raw value)

Standard errors clustered around packages in parentheses for (1), (2), (4) and (5). The variance for (3) and (6) are estimated using the Jackknife variance procedure.

*** p<0.001, ** p<0.01, * p<0.05, + p<0.1

The observations vary between models (1) and (2) because of the unbalanced nature of the data. The observations vary across model (1)-(3) and (4)-(6) because only 1089 python and 1077 R packages out of 1610 packages have commits data on GitHub, which can be associated with just that package.

Table A3. The Effect of GitHub Copilot on the Type of Innovation – Raw Dependent Variable

Dependent Variable	Number of New Commits	
	(1) <i>TWFE</i>	(2) <i>Balanced TWFE</i>
<i>PythonPackage_i X afterCopilot_{it}</i>	0.838** (0.26)	0.40 (0.279)
<i>PythonPackage_i X afterCopilot_{it} X Category Maintenance_i</i>	1.814* (0.76)	1.804* (0.757)
Time Fixed Effect	YES	YES
Package Fixed Effect	YES	YES
# of Obs.	52,856	44,902

Note: This table presents a robustness check where the dependent variable is not logged (i.e., raw value)

Standard errors clustered around packages in parentheses

*** p<0.001, ** p<0.01, * p<0.05 + p<0.1

Table A4. The Effect of GitHub Copilot on the Type of Innovation—Subsample Based on Activity – Raw Dependent Variable

Dependent Variable	Number of New Commits			
	Above Median Package		Below Median Package	
Models	(1) TWFE	(2) Balanced TWFE	(3) TWFE	(4) Balanced TWFE
<i>PythonPackage_i X afterCopilot_t</i>	1.81*** (0.507)	1.27* (0.56)	0.081 (0.056)	0.06 (0.06)
<i>PythonPackage_i X afterCopilot_t X Category Maintenance_j</i>	3.32* (1.508)	3.04* (1.66)	0.56*** (0.013)	0.609*** (0.015)
Time Fixed Effect	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES
# of Obs.	26,072	21,424	26,784	23,478

Note: This table presents a robustness check where the dependent variable is not logged (i.e., raw value)

Standard errors clustered around packages in parentheses

*** p<0.001, ** p<0.01, * p<0.05 + p<0.1

Table A5. The Effect of GitHub Copilot on the Volume of Innovation – Dropping quarters 0,1 and 2

Dependent Variable	Number of New Version Releases			Number of New Commits		
	(1) TWFE	(2) Balanced TWFE	(3) SDiD	(4) TWFE	(5) Balanced TWFE	(6) SDiD
<i>PythonPackage_i X afterCopilot_t</i>	0.194*** (0.015)	0.097*** (0.0151)	0.087*** (0.014)	0.50*** (0.044)	0.374*** (0.046)	0.352*** (0.045)
Time Fixed Effect	YES	YES	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES	YES	YES
# of Obs.	28,619	24,650	24,650	19,930	17,270	17,270

Note: This table presents a robustness check where the first 3 quarters after GitHub Copilot launch are dropped to account for early adopters.

Standard errors clustered around packages in parentheses for (1), (2), (4) and (5). The variance for (3) and (6) are estimated using the Jackknife variance procedure.

*** p<0.001, ** p<0.01, * p<0.05, + p<0.1

The observations vary between models (1) and (2) because of the unbalanced nature of the data. The observations vary across model (1)-(3) and (4)-(6) because only 1089 python and 1077 R packages out of 1610 packages have commits data on GitHub, which can be associated with just that package.

Table A6. The Effect of GitHub Copilot on the Type of Innovation – Dropping quarters 0,1 and 2

Dependent Variable	Number of New Commits	
	(1) TWFE	(2) Balanced TWFE
<i>PythonPackage_i X afterCopilot_t</i>	0.189*** (0.027)	0.12*** (0.03)
<i>PythonPackage_i X afterCopilot_t X Category Maintenance_j</i>	0.154*** (0.024)	0.13*** (0.03)
Time Fixed Effect	YES	YES
Package Fixed Effect	YES	YES
# of Obs.	39,860	34,540

Note: This table presents a robustness check where the first 3 quarters after GitHub Copilot launch are dropped to account for early adopters.

Standard errors clustered around packages in parentheses

*** p<0.001, ** p<0.01, * p<0.05 + p<0.1

**Table A7. The Effect of GitHub Copilot on the Type of Innovation—Subsample Based on Activity -
– Dropping quarters 0,1 and 2**

Dependent Variable	Number of New Commits			
	Above Median Package		Below Median Package	
Models	(1) TWFE	(2) Balanced TWFE	(3) TWFE	(4) Balanced TWFE
$PythonPackage_i X afterCopilot_t$	0.37*** (0.047)	0.317*** (0.051)	0.028 (0.021)	0.15 (0.023)
$PythonPackage_i X afterCopilot_t$ $X Category Maintenance_j$	0.23*** (0.041)	0.189*** (0.046)	0.077* (0.02)	0.092* (0.030)
Time Fixed Effect	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES
# of Obs.	19,592	16,480	20,268	18,060

Note: This table presents a robustness check where the first 3 quarters after GitHub Copilot launch are dropped to account for early adopters.

Standard errors clustered around packages in parentheses

*** p<0.001, ** p<0.01, * p<0.05 + p<0.1

**Table A8. The Effect of GitHub Copilot on the Volume of Innovation-Advancing the treatment
beginning to June 2021**

Dependent Variable	Number of New Version Releases			Number of New Commits		
	(1) TWFE	(2) Balanced TWFE	(3) SDiD	(4) TWFE	(5) Balanced TWFE	(6) SDiD
$PythonPackage_i X afterCopilot_t$	0.164*** (0.012)	0.091*** (0.012)	0.087*** (0.012)	0.417*** (0.035)	0.314*** (0.037)	0.294*** (0.036)
Time Fixed Effect	YES	YES	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES	YES	YES
# of Obs.	38,279	32,045	32,045	26,428	22,451	22,451

Note: This table presents a robustness check where t=0 is advanced from October 2021 to June 2021, to account for technical previews.

Standard errors clustered around packages in parentheses for (1), (2), (4) and (5). The variance for (3) and (6) are estimated using the Jackknife variance procedure.

*** p<0.001, ** p<0.01, * p<0.05, + p<0.1

The observations vary between models (1) and (2) because of the unbalanced nature of the data. The observations vary across model (1)-(3) and (4)-(6) because only 1089 python and 1077 R packages out of 1610 packages have commits data on GitHub, which can be associated with just that package.

Table A9. The Effect of GitHub Copilot on the Type of Innovation—Advancing the treatment beginning to June 2021

Dependent Variable	Number of New Commits	
	(1)	(2)
Models		
	<i>TWFE</i>	<i>Balanced TWFE</i>
<i>PythonPackage_i X afterCopilot_t</i>	0.155*** (0.023)	0.101*** (0.024)
<i>PythonPackage_i X afterCopilot_t</i> <i>X Category Maintenance_j</i>	0.141*** (0.018)	0.126*** (0.020)
Time Fixed Effect	YES	YES
Package Fixed Effect	YES	YES
# of Obs.	52,856	44,902

Note: This table presents a robustness check where t=0 is advanced from October 2021 to June 2021, to account for technical previews. Standard errors clustered around packages in parentheses
 *** p<0.001, ** p<0.01, * p<0.05 + p<0.1

Table A10. The Effect of GitHub Copilot on the Type of Innovation—Subsample Based on Activity—Advancing the treatment beginning to June 2021

Dependent Variable	Number of New Commits			
	Above Median Package		Below Median Package	
	(1)	(2)	(3)	(4)
Models				
	<i>TWFE</i>	<i>Balanced TWFE</i>	<i>TWFE</i>	<i>Balanced TWFE</i>
<i>PythonPackage_i X afterCopilot_t</i>	0.307*** (0.039)	0.248*** (0.042)	0.028 (0.017)	0.022 (0.019)
<i>PythonPackage_i X afterCopilot_t</i> <i>X Category Maintenance_j</i>	0.199*** (0.031)	0.164*** (0.035)	0.085*** (0.02)	0.095*** (0.022)
Time Fixed Effect	YES	YES	YES	YES
Package Fixed Effect	YES	YES	YES	YES
# of Obs.	26,072	21,424	26,784	23,478

Note: This table presents a robustness check where t=0 is advanced from October 2021 to June 2021, to account for technical previews. Standard errors clustered around packages in parentheses
 *** p<0.001, ** p<0.01, * p<0.05 + p<0.1

Table A11. The Effect of GitHub Copilot – Rollbacks as Dependent Variable

Dependent Variable	Rollbacks		
	(4)	(5)	(6)
Models			
	<i>TWFE</i>	<i>Balanced TWFE</i>	<i>SDiD</i>
<i>PythonPackage_i X afterCopilot_t</i>	-0.001 (0.007)	-0.003 (0.008)	-0.002 (0.008)
Time Fixed Effect	YES	YES	YES
Package Fixed Effect	YES	YES	YES
# of Obs.	26,428	22,451	22,451

Note: Standard errors clustered around packages in parentheses for (1), (2), (4) and (5). The variance for (3) and (6) are estimated using the Jackknife variance procedure.
 *** p<0.001, ** p<0.01, * p<0.05, + p<0.1

The observations vary between models (1) and (2) because of the unbalanced nature of the data. The observations vary across model (1)-(3) and (4)-(6) because only 1089 python and 1077 R packages out of 1610 packages have commits data on GitHub, which can be associated with just that package.

Appendix B

The primary task is to determine the category of commits made to GitHub using the commit “comments”.

The box below presents the annotation guidelines and the prompt that we used in employing LLMs for this classification task. The prompt is as follows:

You are an experienced programmer with extensive contributions to Python and R projects. You will receive the name of a GitHub repository along with one of its commit comments. Your task is to meticulously analyze the commit comment and classify it into the most appropriate category based on its purpose or content. The categories are:

1. Code Development: Includes 'Feature Addition' (adding new capabilities), 'Code Optimization' (enhancing performance or efficiency), 'Algorithm implementation' (changing or adding new algorithms).
2. Maintenance: Encompasses 'Bug Fix' (correcting errors), 'Code Cleanup' (improving readability or structure without changing functionality), and 'Dependency Update' (chores and grunt tasks for updating libraries or dependencies), 'Refactoring' (altering code structure without changing behavior), 'Feature Removal or Deprecation' (taking out a feature from the codebase or marking it for future removal)
3. Documentation and Style: Combines 'Documentation' (creating or updating documentation), 'Style Update' (modifying code styling for consistency or readability), new spoken language support and changelog.
4. Testing and Quality Assurance: Involves 'Testing' (adding or updating tests), 'Version Update' (updating version numbers or managing releases) and logging (results log, test log etc)
5. Other: Any changes that do not fit into the above categories (such as license and copyright updates, continuous integration continuous deployment).

Additionally, assess the significance of the change as 'Minor', 'Medium', or 'Significant', and provide an estimated count of changes made. Include a concise justification for your classification.

Your response should be in a valid JSON format and strictly contain only the JSON, with all string values enclosed in double quotes. The response must include 'category' (a string), 'significance' (a string), 'count' (an integer, or a string if the count is unknown), and 'reason' (a string). For example:

```
{
  "category": "Code Development",
  "significance": "Medium",
  "count": 5,
  "reason": "This commit adds a new feature improving data processing efficiency."
}
```

Remember to consider the context and implications of each commit comment while categorizing and assessing its impact.

To choose the best model, we benchmarked some of their performances against human tagging. We first randomly selected 200 Python commit comments and 200 R commit comments and got them tagged by three expert human annotators. We then compared the outcome from 3 different LLMs (GPT 4, GPT 3.5

Turbo, LLaMa 3 70B) with human tags and found that GPT4 and GPT3.5 Turbo outperformed the opensource LLM and were very close to each other. Table B1 shows the outcomes of the model and the Cohen’s Kappa for the agreement between expert human annotators and the LLM predictions. The Cohen’s Kappa is above 0.6 for the GPTs, which puts the agreement at the “substantial” level, as noted in extant research (e.g., Cohen 1960, McHugh 2012). For cost, time and efficiency reasons, we employed GPT 3.5 Turbo for our annotation task. Table B2 presents the model details.

Table B1. The accuracy of LLMs compared to human annotated ground truth.

Model	Agreement Rate	Cohen’s Kappa
gpt-4-1106-preview	85.25%	0.80
gpt-3.5-turbo-0125	81.00%	0.74
LLaMa 3 (70B)	58.00%	0.44

Table B2. The Model Parameters

Model	endpoint	temperature	top_p	frequency_ penalty	presence_ penalty
gpt-3.5-turbo-0125	chat	0.0	1.0	0.0	0.0

Appendix C

In this appendix, we undertake an endeavor to categorize Python and R packages into various domains (e.g., Machine Learning, Artificial Intelligence, Natural Language Processing, Web Development, Networking, and APIs). The categorization ensures that when the Python and R packages are matched for causal analysis, they matched pairs are broadly similar in their utility and use-cases, despite being from different programming languages. In essence, this appendix ensures the robustness of the parametric matching technique used in the TWFE analysis in the paper.

Section C1: Annotation using a Large Language Model

This section outlines the text mining process, leveraging LLMs as expert annotators. The primary challenge in this task lies in the absence of a uniform classification system across programming languages. To overcome this challenge, we use the package descriptions from their respective official repositories: PyPI for Python and CRAN for R. Although concise, these descriptions encapsulate crucial information about the packages' primary functionalities and use-cases, enabling us to utilize LLMs for effective categorization. For example, the package “matplotlib” is described as a “Python plotting package,” whereas ggplot2 is described as a tool to “Create Elegant Data Visualizations Using the Grammar of Graphics” in R language; both of these packages have multiple overlaps in functionalities and utilities.

To establish a comprehensive and mutually exclusive categorization framework for the Python packages, we leveraged the ‘Awesome Python’ list, a curated collection of 646 Python packages categorized into 91 domains on the day of data collection in October 2023. Recognizing that several of these domains had considerable overlap, we employed a combination of LLMs and expertise of a human research assistant well-versed in programming to carefully consolidate these categories. Such a process of careful consolidation of related categories helps in simplifying the annotation process for humans while enhancing the agreeability of machine learning models (Wang et al. 2020, Warrens 2010).

We divided the 646 packages listed in Awesome Python into 2 groups: A randomly chosen set of 517 packages (i.e., 80% of the sample) used for constructing the consolidated categories and the remaining 129

packages as “test” data to ensure the accurate consolidation of domains into categories. This process ensures the creation of a robust taxonomy that encompasses the diverse functionalities of Python packages while maintaining clarity and distinction between categories. As a result of this process, we identified 8 distinct categories for the packages: (1) Data Processing, Analysis and Computing, (2) Machine Learning, Artificial Intelligence, and Natural Language Processing, (3) Data Visualization and Geospatial Analysis, (4) Web Development, Networking, and API calls, (5) Data Storage, Retrieval, and Web Scraping, (6) Security and Cryptography, (7) Testing, Validation, and Code Quality, (8) System Administration, DevOps, and User Experience.

The second step involves creating specific annotation guidelines for use with LLMs. In this step, we follow the annotation guidelines outlined by Lee et al. (2023) and developed the following annotation guidelines for LLMs:

You are tasked with classifying a Python or R package into one of the following aggregated categories based on its description. Each aggregated category corresponds to a specific set of functionalities commonly seen in these languages.

The available aggregated categories are:

1. **Data Processing, Analysis, and Computing:** Includes packages for data cleaning, ETL (Extract, Transform, Load), data validation, data analysis, data formatting, algorithms and design patterns, packages used for scientific research, numerical computation, simulations, statistical analysis, probability, and regression models.
2. **Machine Learning, Artificial Intelligence, and Natural Language Processing:** Includes packages for machine learning, deep learning, and AI tasks such as classification, clustering, model training, processing text data and natural language tasks.
3. **Data Visualization and Geospatial Analysis:** Includes packages used for creating visual representations of data, such as graphs, charts, maps, geographic data, GIS, and mapping.
4. **Web Development, Networking, and API calls:** Includes packages for building web applications, APIs, and servers. Also, includes packages for working with networks and APIs, including HTTP clients and web services.
5. **Data Storage, Retrieval, and Web Scraping:** Includes packages related to database interactions, ORM (Object-Relational Mapping), and data storage. Also include Web Scraping, packages used for extracting data from websites.
6. **Security and Cryptography:** Includes packages focused on security measures such as encryption, authentication, and authorization.

7. Testing, Validation, and Code Quality: Includes packages for software testing, code analysis, code validation, and code debugging.

8. System Administration, DevOps, and User Experience: Includes packages for system administration and monitoring tasks, often related to infrastructure management. System Utilities includes packages for system-level tasks such as command-line tools, job scheduling, and system monitoring, building user interfaces and improving user experience.

Please use the following reasoning process (chain of thought) to classify the package:

1. Analyze the package description.
2. Identify the key functionalities and tasks the package aims to address.
3. Match these functionalities with the aggregated category descriptions provided above.
4. Finally, classify the package into one of the aggregated categories.

Return the result in the following JSON format:

```
{
  "reasoning": "The package focuses on processing images and creating visual outputs. Since its primary function revolves around visual representation of data, it fits into the Data Visualization category.",
  "classification": "Data Visualization and Geospatial Analysis"}
```

The third step involves benchmarking the performance of LLMs against the “ground truth”. Fortunately, the ground-truth is already available through Awesome Python’s categorized packages. Specifically, we utilized the ground truth of the 129 test packages from the above step, which were not involved in generating the annotation guidelines, to benchmark the performance of the LLMs. Table C1 presents the benchmarking results. All the GPT models achieved a Cohen’s Kappa of over 0.6, indicating “substantial agreement” as noted by extant research across discipline (e.g., Cohen 1960, McHugh 2012). Furthermore, as shown in Table C1, OpenAI’s gpt4o-mini performs as well as gpt4o and outperforms GPT4 Turbo. Therefore, we selected gpt4o-mini for classification due to its cost and time efficiencies.

Model	Agreement Rate	Cohen’s Kappa
gpt-4-turbo-2024-04-09	67.44%	0.62
gpt-4o-2024-08-06	71.32%	0.66
gpt-4o-mini-2024-07-18	69.77%	0.64

Table C1. The accuracy of LLMs compared to the ground truth from Awesome Python

Table C2. The Model Parameters

Model	endpoint	temperature	top_p	frequency_penalty	presence_penalty
gpt-4o-mini-2024-07-18	chat	0.0	1.0	0.0	0.0

The final step is to employ gpt4o-mini to tag the all the packages in our dataset into 8 categories. The model parameters for the classification are in Table C2.

Section C2: Propensity Score Matching and DiD Involving Categories

As a supplementary extension of our main analysis, we refine the propensity score matching procedure to incorporate the newly constructed ‘category’ variable from section C1, which categorizes each package. This ensures that the matched treatment and control packages not only share similar pre-treatment characteristics but also belong to the same functional domain. We implement an exact match on ‘category’, guaranteeing that each Python package is paired with an R package from the same category. Due to the stricter matching criteria, not all treatment packages were able to find a suitable match. Consequently, our final dataset for this extended analysis comprises 1,355 unique Python packages (treatment packages) and 1,355 unique R packages (control packages).

The DiD analysis follows the same methodology as described in the paper, employing this newly matched dataset to estimate the impact of GitHub Copilot on open-source packages. Figure C1 displays the event-study coefficients, demonstrating that the parallel trends assumption holds after incorporating the ‘category’ variable in matching (Autor 2003).

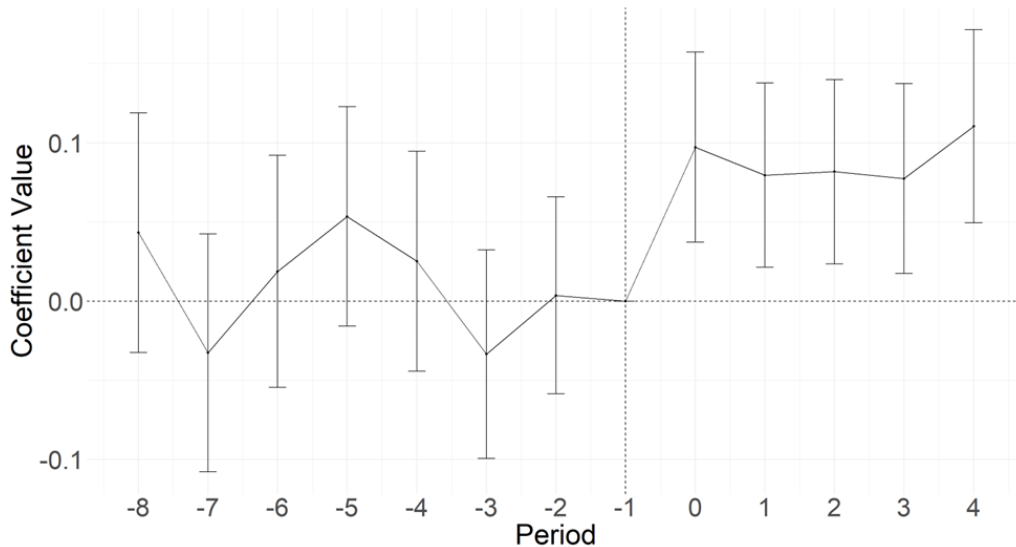


Figure C1. Parallel Trends Graph. (Confidence Interval at 99%)
Note: The y-axis is the log transformation of the new version released each quarter

Table C3 presents the results of our analysis on the impact of LLMs on open-source innovation volume, mirroring the main findings. Consistent with our primary specifications, we observe a significant increase in the number of updates following the release of GitHub Copilot, highlighting the positive impact of LLMs on open-source development activity.

Table C3. The Effect on the Volume of Innovation

Dependent Variable	Number of New Versions		
	(1) <i>TWFE</i>	(2) <i>Balanced TWFE</i>	(3) <i>SynthDiD</i>
Models			
<i>PythonPackage_i X afterCopilot_t</i>	0.08*** (0.013)	0.026* (0.0136)	0.023* (0.013)
Time Fixed Effect	YES	YES	YES
Package Fixed Effect	YES	YES	YES
# of Obs.	32,006	26,169	26,169

Note: *** p<0.001, ** p<0.01, * p<0.05, + p<0.1
Standard Errors clustered around package in parentheses for (1), and (2).
The variance for (3) is estimated using the Jackknife variance procedure.

Appendix References

Cohen J (1960) A coefficient of agreement for nominal scales. *Educational and psychological measurement*. 20(1):37-46.

McHugh ML (2012). Interrater reliability: the kappa statistic. *Biochemia Medica*. 15;22(3):276-82.

Wang Y, He D, Li F, Long X, Zhou Z, Ma J, Wen S (2020) Multi-label classification with label graph superimposing. *Proceedings of the AAAI Conference on Artificial Intelligence* 34(07) pp. 12265-12272).

Warrens MJ (2010) Cohen's kappa can always be increased and decreased by combining categories. *Statistical Methodology*. 7(6):673-7.