It Pays to be the Pilot : The Impact of Generative AI Coding Tools on Developer Productivity*

Christian Wilson†

October, 2024

Abstract

Generative AI tools like ChatGPT have the potential to significantly reshape how individuals work, primarily through automating tasks and increasing worker productivity. Based on research on automation technologies like robots and computers, understanding both which tasks are most likely to be automated and which workers are most likely to see productivity gains is crucial for understanding these technologies' potential impacts on income inequality. I focus on the latter of these two questions and study the introduction of generative AI coding tools on the productivity of open-source software developers. Leveraging these AI coding tools' heterogeneous accuracy across programming languages, I measure these tools' impact on productivity for high and low-skill developers. I find that high-skill developers see the largest changes in productivity in this real-world setting, in contrast to recent findings from laboratory settings and research that suggests that generative AI tools might be "inverse skill-biased". While these tools do increase productivity generally, low-skill programmers are approximately 12.3% less productive than their high-skill counterparts after their introduction.

*Thanks to James Feignbaum and Pascual Restrepo for helpful comments and suggestions. A big thank you to Katie and Rosie for excellent researcher support.

†Boston University. crwilson@bu.edu

1 Introduction

Generative artificial intelligence (AI) is a powerful emerging technology with uncertain impacts on the workforce. Early reports propose that generative AI tools, like ChatGPT, will expose as many as 300 million jobs to automation and potentially automate half of all work activities currently done today (Goldman, 2023; McKinsey, 2023). These reports also forecast generative AI tools to lead to significant GDP growth (as much as $7 trillion globally over a ten-year period) and expect labor productivity to increase by as much as 3.4% annually when combining generative AI tools with other technologies.

Beliefs that generative AI will simultaneously automate jobs and increase productivity lead to a major open question - how might these two major changes impact income inequality? Recent research on automation technologies, such as computers and robots, suggests that these technologies led to a "hollowing out" of the income distribution and an increase in income inequality (Acemoglu and Restrepo, 2022). This occurred through two primary channels - by automating the jobs of often middle-income workers who completed skilled routine tasks, and by increasing the productivity of high-skilled (and more highly paid) workers. These combined effects have led recent automation technologies to be deemed "skill-biased" as they have increased salaries for high-skilled workers but led to no gains (or even decreases) in salaries for less-skilled workers (Katz and Murphy, 1992; Card and Lemieux, 2001; Acemoglu and Autor, 2011).

Recent research on robots and computers provides a roadmap for how to determine whether generative AI is a similarly skill-biased technology, reserach must identify which workers are most exposed to automation and which are likely to see the largest productivity gains. This paper focuses on understanding the latter of the two questions - which individuals are likely to see the largest increases in productivity from using generative AI tools. Early research on this topic in laboratory settings has found that generative AI tools have larger impacts on productivity for those who are less-skilled/experienced (Brynjolfsson et al., 2023; Dell'Acqua et al., 2023; Noy and Zhang, 2023). Recent research in real-world settings, especially software development, has found similar results (Cui et al., 2024; Hoffman et al., 2024; Wiles et al., 2024). These findings have led some to claim that generative AI is in fact "inverse skill-biased", and therefore potentially an engine for decreasing income inequality by raising the relative productivity of lower-skilled workers (Capraro et al., 2024) .

I provide some of the first evidence on the impact of generative AI tools on workers'

marginal product in a real-world setting. Specifically, I analyze the impact of two AI tools, GitHub Copilot and ChatGPT, on the contributions of software developers to open-source projects. I find that these tools do increase developer productivity - developers exposed to Copilot increase their contributions to open-source projects after exposure. These productivity increases do not appear "inverse skill-biased", however. Low-skilled programmers increase their contributions by less on average compared to high-skilled programmers.

Open-source software (OSS) is a highly desirable setting to study the impact of generative AI tools. Firstly, it was one of the earliest applications of generative AI, with GitHub Copilot released to the general public in June of 2022, five months before the general release of ChatGPT. Secondly, OSS development is visible to the public and individual contributions to OSS are tracked and observable, providing measures of individuals' marginal product. Thus, while generative AI adoption is likely still in its infancy, studying OSS provides an early glimpse into the potential differential impact of generative AI tools on worker productivity.

This paper contributes to various literature. The first addresses the impact of Generative AI coding tools on productivity in real world settings. Cui et al. (2024) leverage RCTs employed across three firms and find that programmers given access to GitHub Copilot issue more pull requests than those without access. They also estimate this impact on pull-requests to be greater for newer and less-skilled workers, although these differences are not statistically significant. Hoffman et al. (2024) estimate the impact of GitHub Copilot on maintainer productivity and find that maintainers are more likely to spend time coding and less time on project management activities, like reviewing others' pull requests. Yeverechyahu et al. (2024) leverage the fact that generative AI tend to be much more accurate for Python code versus R. They find that Python projects have more new commits and are more likely to release new versions after the introduction of GitHub Copilot when compared to similar R packages. Wiles et al. (2024) perform an RCT on a group of consultants, finding that consultants with no data science background but with access to ChatGPT are able to complete two of three pre-defined data science tasks. However, they are no more likely to get data science questions correct when access to ChatGPT is taken away post-experiment.

The second branch of literature is that on productivity impacts of generative AI generally and which highlights the heterogenous effects of generative AI tools across workers of different skill levels. Brynjolfsson et al. (2023) find that access to an AI-

based chatbot leads to increases in productivity for customer service agents, with those gains primarily coming from low-skilled workers. Dell'Acqua et al. (2023) find that consultants using AI to complete one of 18 realistic consulting tasks were more productive and produced work of higher quality, with performance improvements relatively stronger for less-skilled consultants. Noy and Zhang (2023) find college students with randomized access to ChatGPT are able to produce writing assignments faster and of higher quality, with ChatGPT driving a decrease in quality differences across students.

The third is on the labor market implications of AI usage. This literature focuses primarily on impacts from earlier AI and machine learning tools. Webb (2020) uses the historical impact of robots and software on employment and wages to predict the impact of artificial intelligence, predicting that AI will reduce 90:10 wage inequality. Acemoglu et al. (2022) finds that firms for which AI tools are beneficial have increased hiring for AI positions and reduced hiring for non-AI positions, with uncertain aggregate labor market implications.

Finally, this paper contributes to the more general literature on the consequences of recent automation technologies. Acemoglu and Restrepo (2022) find that automation has displaced workers specialized in routine tasks, leading to significant changes in wage inequality in the US. Similarly, Acemoglu and Restrepo (2020) find that robots and automation reduce employment and wages in local US labor markets most exposed to these new technologies. Graetz and Michaels (2018) find that increased robot use over the period 1993 to 2007 led to an increase in annual labor productivity group but did reduce low-skilled workers' employment share.

The paper proceeds as follows: Section 2 provides background on LLMs, specifically GitHub Copilot and ChatGPT, as well as background on OSS. Sections 3 and 4 outline my data and methodology. I then present my empirical results in Section 5. Finally, Section 6 discusses the results and Section 7 concludes.

2 Background

OSS, GitHub, and Git

Open-source software (OSS) refers to software where the source code (the code the software is based on) is publicly available, and anyone can view, comment, or modify the software's source code (Opensource.com, 2024). OSS plays a significant role in today's software ecosystem, with one study estimating that OSS can be found in 96% of codebases (Bals, 2024). There exist popular open-source versions of operating systems (Linux), programming languages (Python), and web browsers (Firefox). (Hoffmann et al., 2024) estimates that the demand-side value of OSS (the total cost firms would need to pay to recreate the OSS they currently use) to be $8.8 trillion.

Many open-source projects use GitHub to store their source code and make it available to others. GitHub boasts that it is the "largest open source community in the world" (GitHub, 2024). For reference, GitHub has approximately 20,000 open-source projects with at least 500 stars, meaning at least 500 users are actively following development of the project. GitLab, GitHub's largest competitor, has around 50 of such repositories.

Members of the open-source community are able to view and contribute to open-source GitHub projects using Git, a version control and collaboration tool. In Git, a project is known as a repository. Within a repository, multiple individuals can be working on code at one time. To facilitate this, project creators/owners designate one copy of the code to be the main/primary version which is compiled and released as software at regular intervals by owners. To make modifications, users can duplicate the entire repository by *forking* it - creating an identical replica of the repository, but one they have full control over. Users can also make modification by creating a new *branch* of the project, which lives inside the same repository but is a copy of the main branch a specific point in time. Typically, only project owners, also known as *maintainers*, have access to directly modify the main branch of a repository. Thus, by forking or branching, members of the open-source community have the ability to modify the project's source code freely.

When a project stores its source code on GitHub, the code lives on GitHub servers. Thus, to make changes, users *clone* (copy) the code to their own computer. Users then *commit* changes at regular intervals to codify their modifications. When they ultimately want to update the code that lives on GitHub, users *push* these changes to GitHub. When a user wishes to have their modifications incorporated into the main

branch of a project, they issue a *pull request* to the maintainers of the project. If the maintainers choose to accept the changes, they merge the user's fork or branch into the current version of the main branch.

GitHub Copilot and ChatGPT

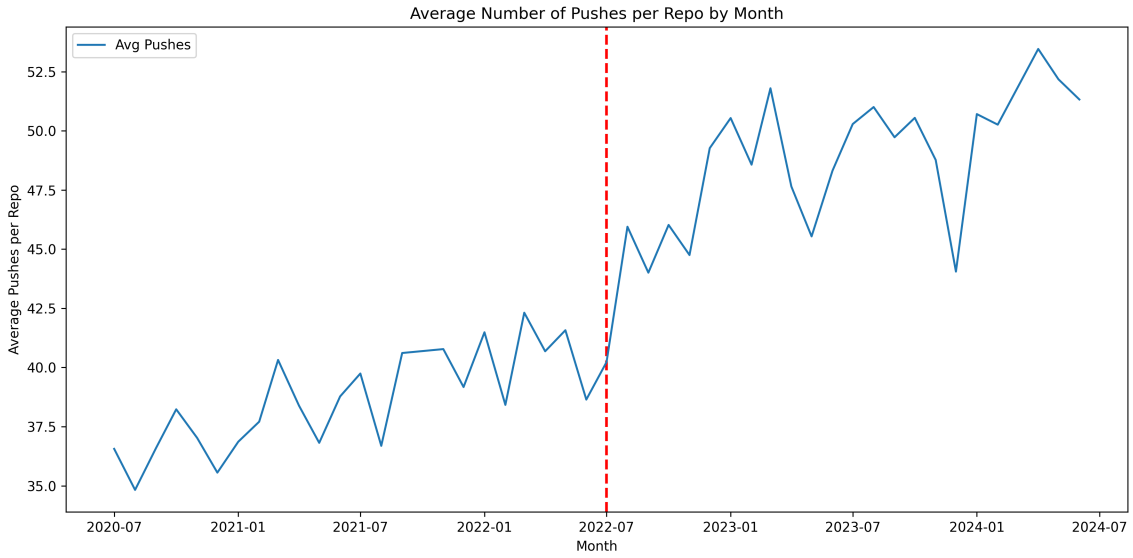
On June 21, 2022, Microsoft made generally available GitHub Copilot, an "AI pair programmer that helps (programmers) write better code". Figure 1 shows the average number of contributions to open source projects per month before and after Copilot is introduced. Copilot leverages a large language model (LLM) called Codex, developed by the nonprofit OpenAI, to suggest new code based on a combination of both the code a programmer had already written and prompts. In an early randomized trial, GitHub researchers found that programmers given access to GitHub Copilot were able to complete a designated task 55% faster than those without access. A qualitative survey on a large group of Copilot users also found that developers reported being generally more productive and faster with repetitive tasks when using Copilot.

On November 30, 2022, Open AI separately released a chat-oriented LLM known as ChatGPT, which was accessible generally via a web interface. ChatGPT integrated a prior LLM developed by OpenAI known as GPT 3.0 with their Codex model, creating a unified LLM that could respond to general text prompts and generate code based on prompts (Zhao et al., 2024). A few months later in March of 2023, OpenAI's ChatGPT interface was updated to use a new model, termed GPT 4.0, which featured superior performance and the ability to respond to much larger prompts. At that time, OpenAI rebranded the original ChatGPT model (and variants) as GPT 3.5. On June 29th, 2023, GitHub Copilot updated their base model from Codex to GPT 3.5. [1]

The availability of these new code generation tools made a significant impact on the developer community. One large survey of developers from March of 2023 reported that 44% of professional developer respondents were already using an AI tool (StackOverflow, 2023). By 2024, 62% of professional developer respondents in a similar survey noted they were using an AI tool as part of their workflow (StackOverflow, 2024). 83% of professional developers in the 2024 survey believed improved productivity was the primary benefit of code generation tools.

[1] While other LLM code generation tools were developed over the sample period, I focus primarily on these two OpenAI-based models due to their superior relative performance.

Figure 1. Average Push Events per Repository per Month



3 Data

GitHub Data

Data on open-source software comes from GH Archive, a project that aims to "record the public GitHub timeline, archive it, and make it easily accessible for further analysis." These data include all public events on GitHub, which include individual developers pushing code to a repository, requesting code to be pulled into a repository via a pull request, individuals users notifying developers of issues and requesting new features, and more.

Project contributions are measured in two ways - pushes and commits. Pushes are instances where a developer submits code to a repository branch, while commits identify specific sets of changes the developer makes to the project codebase. Commits can be of an arbitrary size (Dressen, 2021) and therefore are not necessarily a more precise measure of the number of lines of code a developer is modifying or adding to the repository. Importantly, both measures correlate positively with the amount of code that the developer changed, which I denote to be their contribution to the repository/project. In Section 5, I report my results for pushes only.

GH Archive includes the entire universe of public repositories, and therefore in-

cludes everything from student homework assignments to widely used open-source technologies like Docker and Apache Spark. I proxy for repository quality with the repository's stargazer count (stars), which is equivalent to the number of GitHub users that receive updates about the repository. The intuition is that more popular projects are likely to have more users, and therefore require tighter controls around code review and development in order to avoid bugs that would upset their user base. In my analyses I limit my sample to repositories with at least 50 stars - approximately 2.3% of all repositories in my sample.

In addition to a repository's stars, I also track the primary programming language associated with the repository. Some projects may be multi-lingual, such as web-development projects that utilize both JavaScript and PHP. The primary programming language is determined as the programming language that makes up the largest share of bytes of code in a project.

Similar to Cui et al. (2024), I proxy for a developer's skill level by their experience. In this setting, I define a developer's experience as the amount of time since their first push to a repository. I then categorize developers into two groups - those with three years of experience or less and those with more than three years of experience. This roughly maps to the distinction between junior (L1 and L2) and senior developers (Adams, 2023).

LLM Accuracy Data

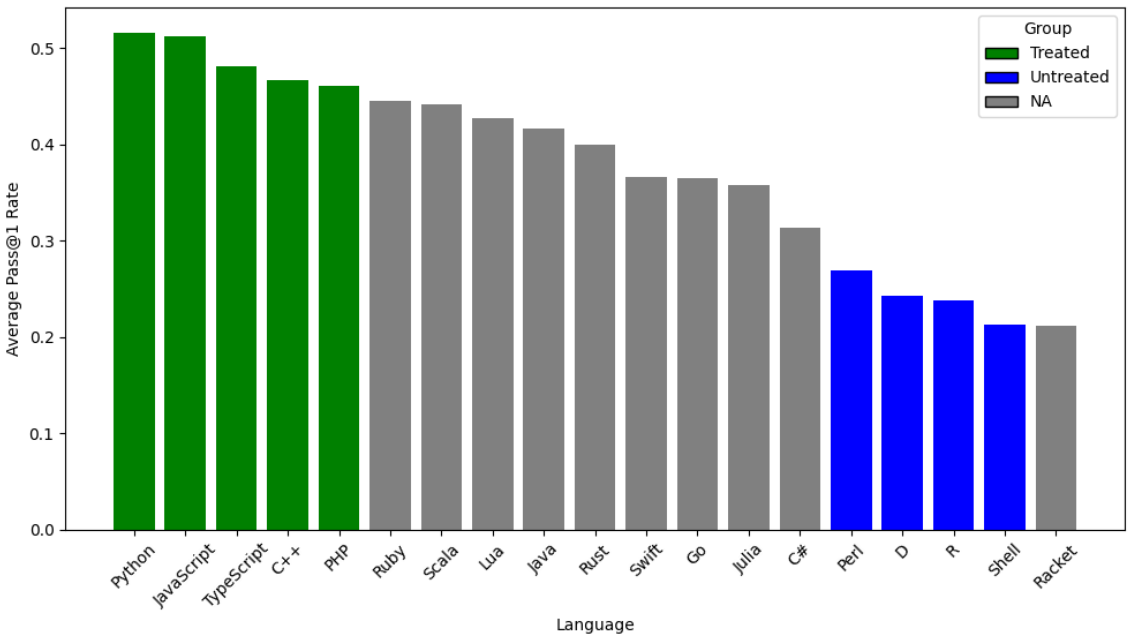
Data on the accuracy of LLM code generation comes from the Multipl-E project (Cassano et al., 2023). MultiPL-E is "a system for translating unit test-driven neural code generation benchmarks to new languages." Many popular benchmarks for evaluating the accuracy of large language models focus on LLMs' ability to generate correct Python code. The Multipl-E project extends two popular Python-based benchmarks, HumanEval and MBPP, to 18 additional programming languages, providing a system for evaluating code-generating LLM performance across languages.

The Multiple-E team evaluate Codex, along with two other code-generating LLMs, against their Multipl-E benchmarks. Specifically, the authors measure the share of benchmark questions a model correctly answers on its first attempt, which they denote as a model's "Pass@1 Rate", and repeat this procedure for all 19 languages. The authors find that Codex performs extremely well across languages (especially in comparison to the other evaluated models), with the LLM able to not only generate

accurate Python code, but accurate code for lesser-used languages like Scala and Rust as well.

I present Codex's Pass@1 rates for all 19 languages averaged across Multiple-E's HumanEval and MBPP benchmarks in Figure 2. Notably, there is significant heterogeneity in Codex's Pass@1 rates, with Codex able to answer Python and JavaScript questions correct about half the time, while it generates accurate R code for less than one quarter of benchmark questions. I leverage this heterogeneity to separate languages into two groups - high and low LLM impact, which I designate as "treated" and "untreated". Languages included in these groups are identified in Figure 2. Treated languages are defined as the four languages for which Codex is most accurate in generating code. Here, I combine JavaScript and TypeScript since they are extremely similar languages, and many JavaScript developers are moving to TypeScript (Baisen, 2022). Untreated languages I define similarly - as Codex's four least accurate languages. Here, I exclude Racket from the untreated group as it makes up a negligible share of repositories/code in the GitHub Archive data, even though it is the language for which Codex is least accurate.

Figure 2. Average Pass@1 Rates by Programming Language



Pass@1 rate denotes the average share of all problems successfully completed by Codex LLM across benchmarks (HumanEval and MBPP).

4 Methodology

Identification of the impact of LLMs on OSS production and OSS developer productivity is challenging in this setting as I don't observe whether individual programmers use Copilot or GPT. Thus, I target a different estimand - the treatment effect of *introducing* LLMs into the OSS community. However, this poses an identification problem as all developers are treated, and all are treated at the same time.

To mitigate concerns that unconditional estimates of the introduction of Copilot (and other LLMs) on productivity may be biased, I leverage the fact these tools are not equally effective at producing accurate code across programming languages. I denote programming languages for which Copilot and ChatGPT (both powered by Open AI Codex) are most accurate as "treated" languages, and those for which it is least accurate as "untreated" (See Section 4 for more details). Appendix A shows the number of average push events for treated and untreated repositories around the introduction of Copilot.

Leveraging these treated and untreated programming languages, I target a slightly more precise and much more identifiable estimand - the incremental treatment effect of moving from a low-accuracy language to a high-accuracy language. Under the assumption that LLMs provide no benefit to programmers who program in low-accuracy languages, this treatment effect would then be equivalent to the impact of the introduction of LLMs on high-accuracy programmers. I will assume that LLMs' impact on low-accuracy language programmers is weakly in the same direction as high-accuracy language programmers, and thus my estimates provide a lower-bound on my estimand of interest. [2]

I estimate two treatment effects of interest - the effect of the introduction of Copilot on repository-level production as well as the effect of the introduction of Copilot on the relative production of low-skilled programmers. In both cases, I measure productivity as pushes per month and take logs so that estimates are interpreted as approximate percent changes.

Repository-level effects I estimate via a standard difference-in-differences, using the following specification:

[2] I am currently working on making this more formal.

$$Log\_Push\_Count_{rt} = \beta_0 + \beta_1 Treated_{rt} + \beta_2 After\_Copilot_t + \beta_3 Treated_{rt} \times After\_Copilot_t + \varepsilon_{rt}$$

where $Log\_Push\_Count_{rt}$ is the log of the number of pushes to repository r in month t, $Treated_{rt}$ is whether the primary languageof the repository is a treated language, and $After\_Copilot_t$ is whether the month is July 2022 or later. β_3 denotes the coefficient of interest. Given concerns that treatment effects may be changing over time (especially given that models are continuously updated and new models are introduced), I also estimate average treatment effects by month, following the procedure of Callaway and Sant'Anna (2021).

Developer-level effects I estimate via a triple difference. Estimation comes from the following specification:

$$\begin{aligned} Log\_Push\_Count_{it} = \beta_0 &+ \beta_1 Treated_{it} + \beta_2 After\_Copilot_t + \beta_3 Young_i \\ &+ \beta_4 Treated_{it} \times After\_Copilot_t \\ &+ \beta_5 Treated_{it} \times Young_i \\ &+ \beta_6 After\_Copilot_t \times Young_i \\ &+ \beta_7 Treated_{it} \times Young_i \times After\_Copilot_t + \varepsilon_{rt} \end{aligned}$$

where $Log\_Push\_Count_{it}$ is the log of developer i's pushes to all repositories in month t, $Treated_{it}$ is whether a deverloper's primary language is a treated language, $After\_Copilot_t$ is whether the month is July 2022 or later, and $Young_i$ is whether the developer had 3 years of experience or less at the time Copilot was introduced. β_7 denotes the coefficient of interest.

5 Results

5.1 Repository-Level Effects

As a baseline estimate for the effect of Copilot on repository-level production, I estimate a simple regression of log push count on a indicator for whether Copilot was available in a given month. The results are presented in Table 1. I find that the introduction of Copilot is associated with an approximately 10% increase in the num-

ber of pushes to a repository in a given month, with and without controlling for programming language.

Table 1. Repository-Level Unconditional Estimates

	Log Push Count	
	OLS	FE
After Copilot	0.101***	0.099***
	(0.002)	(0.002)
Language FE	No	Yes
Observations	2,251,471	2,251,471
R^2	0.001	0.006
Notes:	***Significant at the 1 percent level.	
	**Significant at the 5 percent level.	
	*Significant at the 10 percent level.	

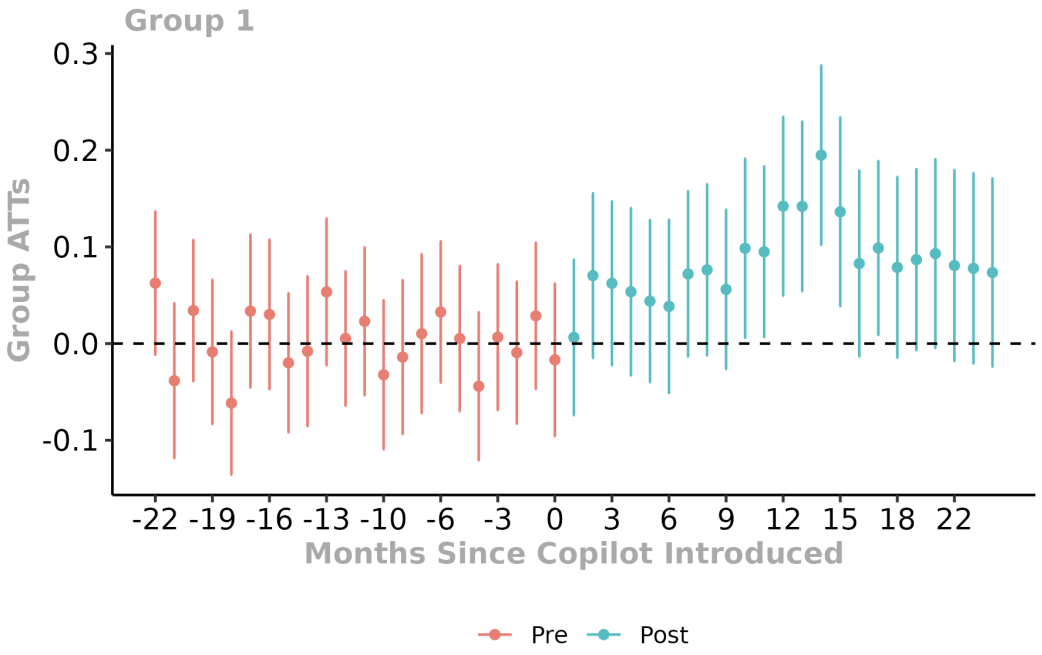
Table 2. Repository Difference in Difference Estimates

	Log Push Count	
	DID	SDID
Copilot ATT	0.100***	0.085***
	(0.008)	(0.020)
VCov	HAC	Bstrap
Observations	2,251,471	2,251,471
Notes:	***Significant at the 1 percent level.	
	**Significant at the 5 percent level.	
	*Significant at the 10 percent level.	
	SDID indicates ATT estimate from Group-Time ATT estimate aggregation.	

The results of Table 1 may suffer from bias if there exist other confounding events around the introduction of Copilot that also impacted code contributions. To account for this, I estimate the effect of Copilot across treated and untreated repositories. The first column of Table 2 presents the results of a difference-in-differences estimation of the effect of Copilot on repository-level production. Comparing across treated and

non-treated repositories, I find a similar 10% increase in the number of pushes to treated repositories in a given month.

Figure 3. Repository Monthly Average Effects



Given the panel structure of the data and the likelihood that the effect of Copilot on repository-level production is changing over time (due to tool uptake and model refinements), I estimate ATTs by month and present thsoe estimates in Figure 3. Monthly estimates indicate that the effect of Copilot peaks approximately one year after it's introduction, stabilizing at around 10 percent thereafter. The second column of Table 4 presents the results of aggregating these ATTs across treated months. I find a slightly smaller ATT of approximately 8.5% when taking into account monthly treatment heterogeneity.

5.2 Developer-Level Effects

Table 3 presents baseline estimates for the effect of Copilot on actor-level production, estimated via a simple regression of log push count on a indicator for whether Copilot was available in a given month. I find that the introduction of Copilot is associated

with an approximately 7.5% increase in the number of pushes to a repository in a given month, or 6.8% when controlling for programming language.

Table 3. Baseline Estimates of the Effect of Copilot

	Log Push Count	
	(1)	(2)
After Copilot	0.075***	0.068***
	(0.002)	(0.002)
Language FE	No	Yes
Observations	2,901,169	2,901,169
R^2	0.001	0.009
Notes:	***Significant at the 1 percent level.	
	**Significant at the 5 percent level.	
	*Significant at the 10 percent level.	

Table 4 presents the results of difference-in-difference estimations for the effect of Copilot on actor-level production. I estimate the difference in difference for 4 seperate groups, defined in Appendix B. Magnitudes of the ATT estimates are declining with group size, which aligns with the fact that groups successively include treated and untreated programming languages with smaller differences in code generation accuracy. Based on the largest group, I find that the introduction of Copilot is associated with an approximately 6.2% increase in the number of pushes, slightly smaller than the unconditional baseline estimates.

I present estimates of the relative impact of Copilot and ChatGPT on low-skilled programmer productivity in Table 5. Low-skilled programmers are those who are first observed on GitHUb less than 3 years before the introduction of GitHub Copilot. Results from the largest comparison group indicate that less-experienced programmers contribute to top repositories 12.3% less than more-experienced programmers after the introduction of Copilot. For the largest comparison group, this 12.3% difference negates the estimated 8.3% increase in pushes for all programmers stemming from the introduction of Copilot. However, programmers are estimated to have productivity increases after Copilot on average, likely due to returns on experience for all young developers (treated and not treated).

Table 4. Difference in Difference Results

	Log Push Count			
	Group 1	Group 2	Group 3	Group 4
After Copilot * Treated	0.136***	0.096***	0.082**	0.072**
	(0.019)	(0.033)	(0.034)	(0.030)
Treated	0.136***	0.117***	0.152***	0.124***
	(0.013)	(0.014)	(0.044)	(0.047)
After Copilot	−0.074***	−0.007	−0.007	0.001
	(0.019)	(0.028)	(0.026)	(0.023)
Constant	1.912***	1.911***	1.913***	1.933***
	(0.013)	(0.001)	(0.003)	(0.026)
VCov	HAC	CL	CL	CL
Observations	648,955	1,614,640	1,941,449	2,096,379
R^2	0.001	0.001	0.001	0.001

Notes:

***Significant at the 1 percent level.
**Significant at the 5 percent level.
*Significant at the 10 percent level.

See Appendix A for group definitions.
Clustered SEs clustered at language level.

Table 5. Triple Difference Results

	Log Push Count			
	Group 1	Group 2	Group 3	Group 4
After Copilot * Treated * Young	−0.100	−0.124***	−0.134***	−0.123***
	(0.070)	(0.019)	(0.017)	(0.018)
Treated	0.137***	0.130***	0.164***	0.132***
	(0.014)	(0.016)	(0.045)	(0.050)
After Copilot	−0.084***	−0.019	−0.019	−0.010
	(0.020)	(0.027)	(0.026)	(0.022)
Young	−0.184***	−0.004	−0.007	−0.042
	(0.046)	(0.054)	(0.052)	(0.076)
After Copilot * Treated	0.142***	0.108***	0.094***	0.083***
	(0.020)	(0.034)	(0.034)	(0.031)
Treated * Young	0.044	−0.128**	−0.125**	−0.088
	(0.047)	(0.054)	(0.053)	(0.076)
After Copilot * Young	0.136**	0.133***	0.138***	0.131***
	(0.069)	(0.003)	(0.007)	(0.009)
Constant	1.926***	1.911***	1.914***	1.937***
	(0.013)	(0.006)	(0.007)	(0.032)
Observations	648,955	1,614,640	1,941,449	2,096,379
R^2	0.002	0.002	0.002	0.002

Notes:

***Significant at the 1 percent level.
**Significant at the 5 percent level.
*Significant at the 10 percent level.

See Appendix A for group definitions.
Clustered SEs clustered at language level.

6 Discussion

Developer and repository-level estimates suggest that access to LLMs has increased the rate at which developers contribute code to public repositories. These repository-level findings align with recent papers that indicate individual productivity gains from LLMs translate to increased productivity for real-world employees. In the open-source developer setting, this result implies that more productive developers are able to find additional tasks to complete, rather than simply completing a defined set of tasks more quickly; a crucial step in translating productivity gains from LLMs into increased economic output. However, additional research is needed to further measure the impact of these additional raw lines of code on more economic objects, such as the amount of code integrated into project codebases (merged pull requests) and software releases.

Developer-level estimates on the productivity of low and high-skilled programmers suggest that increases in open-source code for more popular repositories come primarily from more experienced programmers. Thus, while prior research has found that LLMs have a relatively greater impact of the capabilities of low-skilled programmers, these findings do not seem to translate to relative increases in contributions to OSS.

There are multiple possible explanations for why LLMs' relatively greater impact on low-skilled programmer capabilities may not translate into a realtively greater impact on code developement. One is that LLMs can generate incorrect code, and since low-skilled programmers may not have the ability to discern accuracy of generated code, they are not able to trust the results. Low-skilled programmers then can't use AI-generated code to solve additional tasks that are outside the scope of their knowledge. Alternatively, LLMs may help low-skilled programmers complete tasks more efficiently, but these programmers may not be able to generate ideas for additional tasks to complete. High-skill programmers, on the other hand, are likely able to quickly gauge the accuracy of generated code and find new tasks to complete. Importantly, regardless of mechanism, the absence of relative productivity increases for low-skilled programmers in this real-world setting implies that LLMs are not or are not yet engines for "inverse skill-biased" technological change. Additional research is needed to understand why increases in individual capabilities might not translate to increased productivity, as well as how this might change as LLMs become more accurate at generating code.

7 Conclusion

I study the impact of generative AI tools on high and low-skilled worker productivity in a real-world setting - open source software development. Leveraging the heterogeneous accuracy of generative AI tools across programming languages, While I find these tools do make workers more productive generally, I find that low-skilled workers are approximately 12.3% less productive than their high-skilled counterparts after their introduction. This finding is in contrast with recent research suggesting that generative AI tools are "inverse skill-biased" and emphasizes the need for additional research in order to better understand the generative AI's potential impacts on income inequality.

References

Acemoglu, Daron and David Autor, "Skills, Tasks and Technologies: Implications for Employment and Earnings," *Handbook of Labor Economics*, 2011.

_ **and Pascual Restrepo**, "Robots and Jobs: Evidence from US Labor Markets," *Journal of Political Economy*, 2020, *128* (6), 2188–2244.

_ **and** _ , "Tasks, Automation, and the Rise in U.S. Wage Inequality," *Econometrica*, 2022, *90* (5), 1973–2016.

_ **, David Autor, Jonathon Hazell, and Pascual Restrepo**, "Artificial Intelligence and Jobs: Evidence from Online Vacancies," *Journal of Labor Economics*, 4 2022, *40*.

Adams, Todd, "Leveling Up: Defining the Ladder of Software Engineer Levels - Terminal.io — terminal.io," https://www.terminal.io/engineers/blog/defining-the-ladder-of-software-engineer-levels October 19 2023.

Baisen, Andrew, "Moving from JavaScript to TypeScript," https://dev.to/andrewbaisden/moving-from-javascript-to-typescript-40ac 2022.

Bals, F., "2024 Open Source Security and Risk Analysis Report (OSSRA): Black Duck," February 27 2024.

Brynjolfsson, Erik, Danielle Li, and Lindsey Raymond, "Generative AI at Work," 2023.

Callaway, Brantly and Pedro H.C. Sant'Anna, "Difference-in-Differences with multiple time periods," *Journal of Econometrics*, 2021, *225* (2), 200–230. Themed Issue: Treatment Effect 1.

Capraro, Valerio, Austin Lentsch, Daron Acemoglu, Selin Akgun, Aisel Akhmedova, Ennio Bilancini, Jean-François Bonnefon, Pablo Brañas-Garza, Luigi Butera, Karen M Douglas, Jim A C Everett, Gerd Gigerenzer, Christine Greenhow, Daniel A Hashimoto, Julianne Holt-Lunstad, Jolanda Jetten, Simon Johnson, Werner H Kunz, Chiara Longoni, Pete Lunn, Simone Natale, Stefanie Paluch, Iyad Rahwan, Neil Selwyn, Vivek Singh, Siddharth Suri, Jennifer Sutcliffe, Joe Tomlinson, Sander van der Linden, Paul A M Van Lange, Friederike Wall, Jay J Van Bavel, and Riccardo Viale, "The impact of generative artificial intelligence on socioeconomic inequalities and policy making," *PNAS Nexus*, 06 2024, *3* (6), pgae191.

Card, David and Thomas Lemieux, "Can Falling Supply Explain the Rising Return to College for Younger Men? A Cohort-Based Analysis*," *The Quarterly Journal of Economics*, 05 2001, *116* (2), 705–746.

Cassano, Federico, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda, "MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation," *IEEE Transactions on Software Engineering*, 2023, *49* (7), 3675–3691.

Cui, Zheyuan, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz, "The Effects of Generative AI on High Skilled Work: Evidence from Three Field Experiments with Software Developers," Working Paper, http://dx.doi.org/10.2139/ssrn.4945566 September 2024.

Dell'Acqua, Fabrizio, Edward McFowland, Ethan R. Mollick, Hila Lifshitz-Assaf, Katherine Kellogg, Saran Rajendran, Lisa Krayer, François Candelon, and Karim R. Lakhani, "Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality," *SSRN Electronic Journal*, 2023.

Dressen, Austin, "Engineering Metric: Commit Size — Allstacks," https://www.allstacks.com/blog/engineering-metric-commit-size July 8 2021.

GitHub, "GitHub Open Source," https://github.com/open-source 2024.

Goldman, "Generative AI could raise global GDP by 7%," https://www.goldmansachs.com/insights/articles/generative-ai-could-raise-global-gdp-by-7-percent April 5 2023.

Graetz, Georg and Guy Michaels, "Robots at Work," *The Review of Economics and Statistics*, 12 2018, *100* (5), 753–768.

Hoffman, Manuel, Sam Boysel, Frank Nagle, Sida Peng, and Kevin Xu, "Generative AI and Distributed Work : Evidence from Open Source Software," Working Paper 207681, National Bureau of Economic Research 2024.

Hoffmann, Manuel, Frank Nagle, and Yanuo Zhou, "The Value of Open Source Software," Working Paper, Harvard Business School Strategy Unit, http://dx.doi.org/10.2139/ssrn.4693148 January 2024.

Katz, Lawrence F. and Kevin M. Murphy, "Changes in Relative Wages, 1963–1987: Supply and Demand Factors*," *The Quarterly Journal of Economics*, 02 1992, *107* (1), 35–78.

McKinsey, "The economic potential of generative AI: The next productivity frontier," https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#introduction June 14 2023.

Noy, Shakked and Whitney Zhang, "Experimental evidence on the productivity effects of generative artificial intelligence," *Science*, 7 2023, *381*.

Opensource.com, "What is open source?," https://opensource.com/resources/what-open-source 2024.

StackOverflow, "Stack Overflow Developer Survey 2023," 2023.

_ , "Stack Overflow Developer Survey 2024," 2024.

Webb, Michael, "The Impact of Artificial Intelligence on the Labor Market," Working Paper, Stanford University 2020.

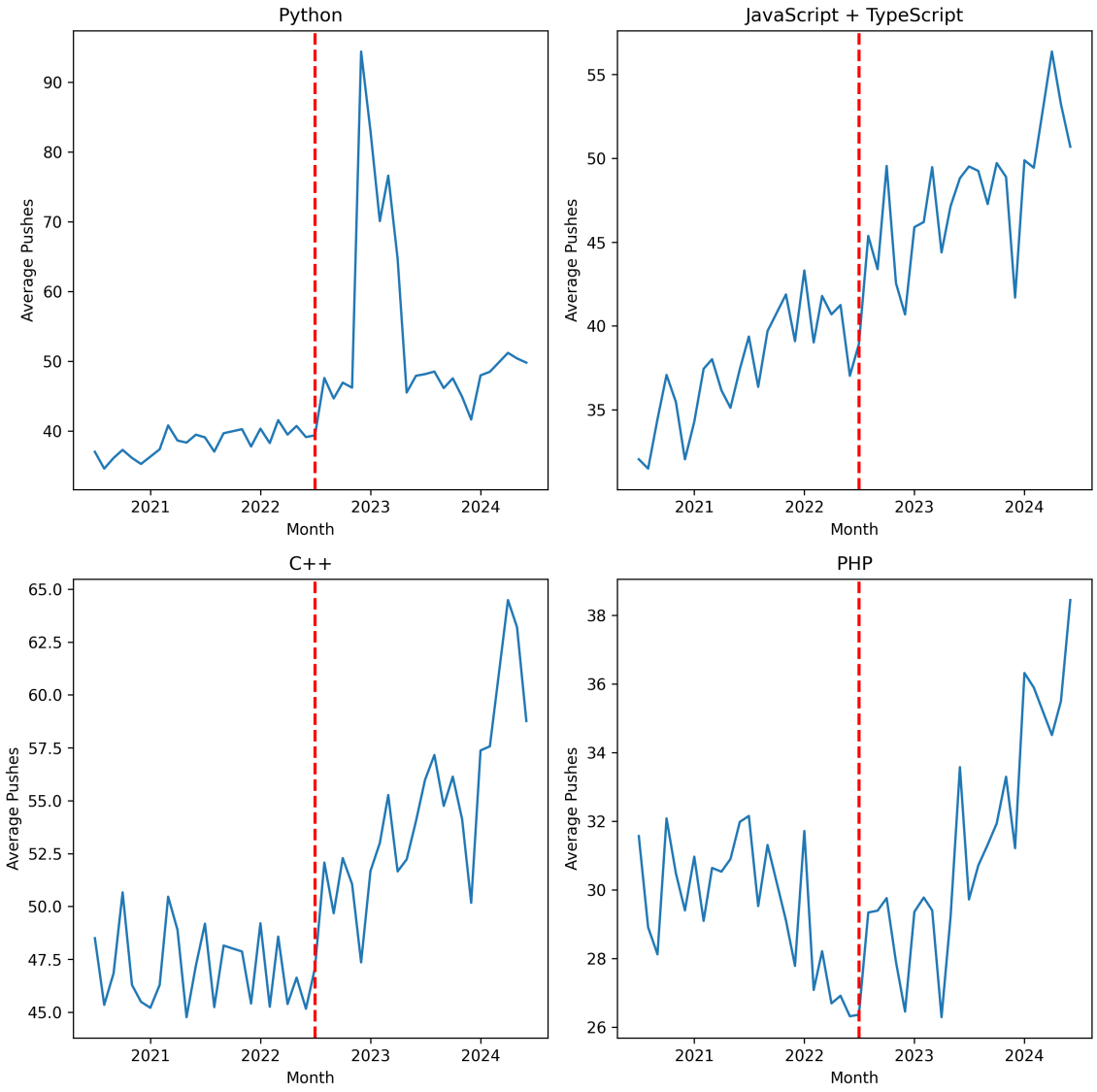
Wiles, Emma, Lisa Krayer, Mohamed Abbadi, Urvi Awasthi, Ryan Kennedy, Pamela Mishkin, Daniel Sack, and François Candelon, "GenAI as an Exoskeleton: Experimental Evidence on Knowledge Workers Using GenAI on New Skills," Working Paper, http://dx.doi.org/10.2139/ssrn.4684662 September 2024.

Yeverechyahu, Doron, Raveesh Mayya, and Gal Oestreicher-Singer, "The Impact of Large Language Models on Open-source Innovation: Evidence from GitHub Copilot," Working Paper, http://dx.doi.org/10.2139/ssrn.4684662 September 2024.

Zhao, Wayne Xin, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen, "A Survey of Large Language Models," 2024.

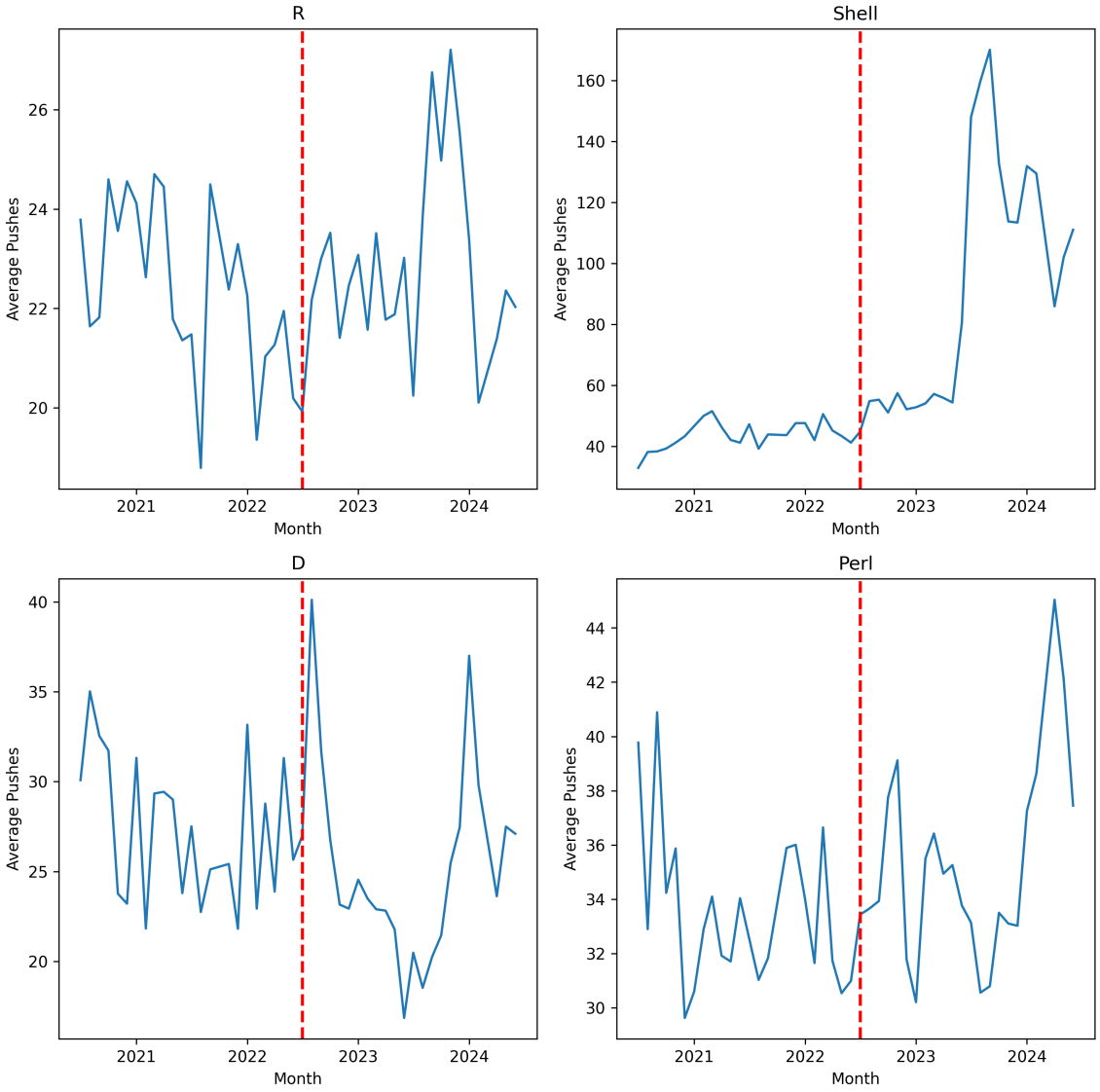
A Average Push Events by Treatment Group

Appendix Figure A1. Pushes per Repo per Month - Treated Languages



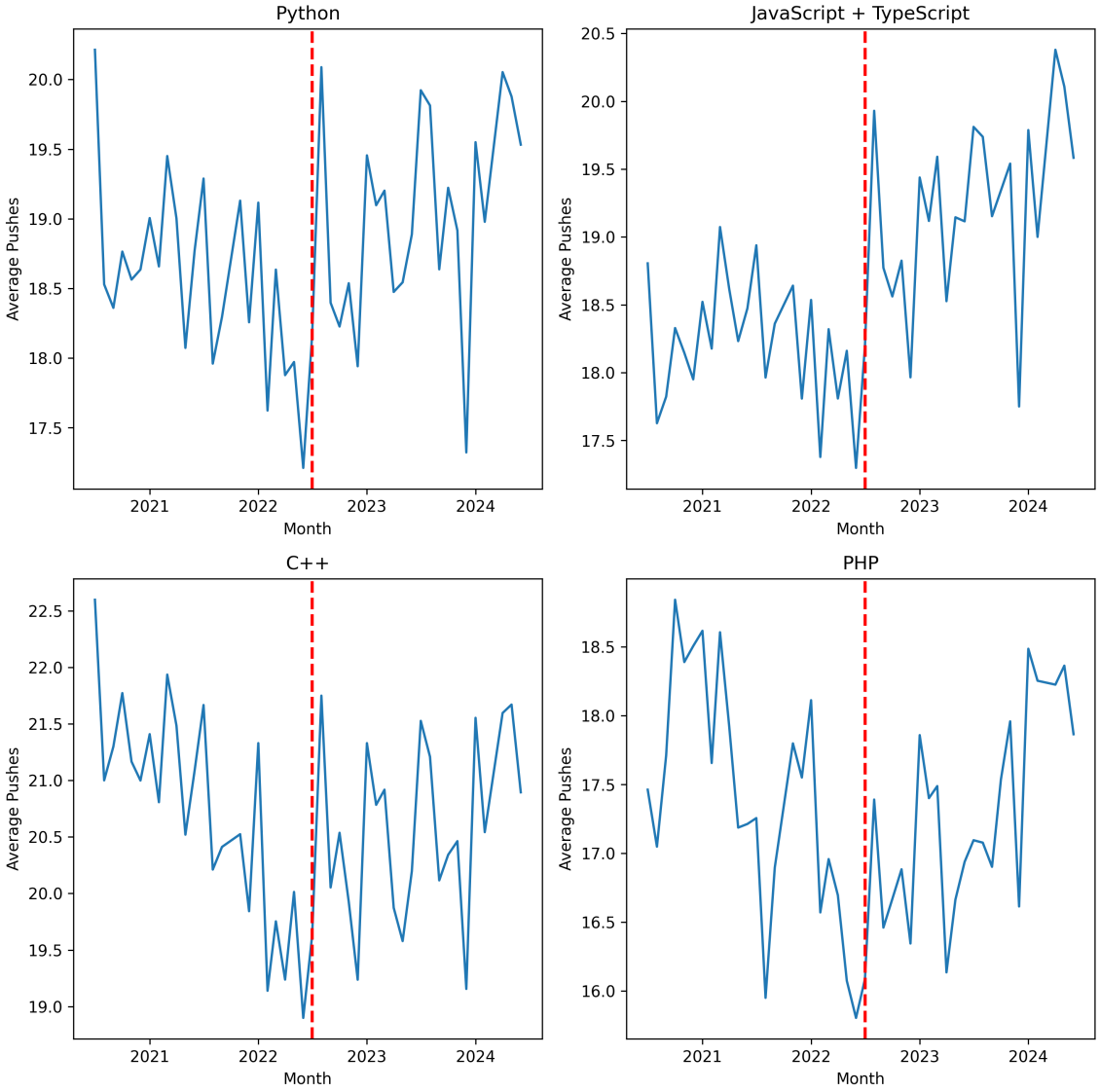
This figure plots . . .

Appendix Figure A2. Pushes per Repo per Month - Untreated Languages



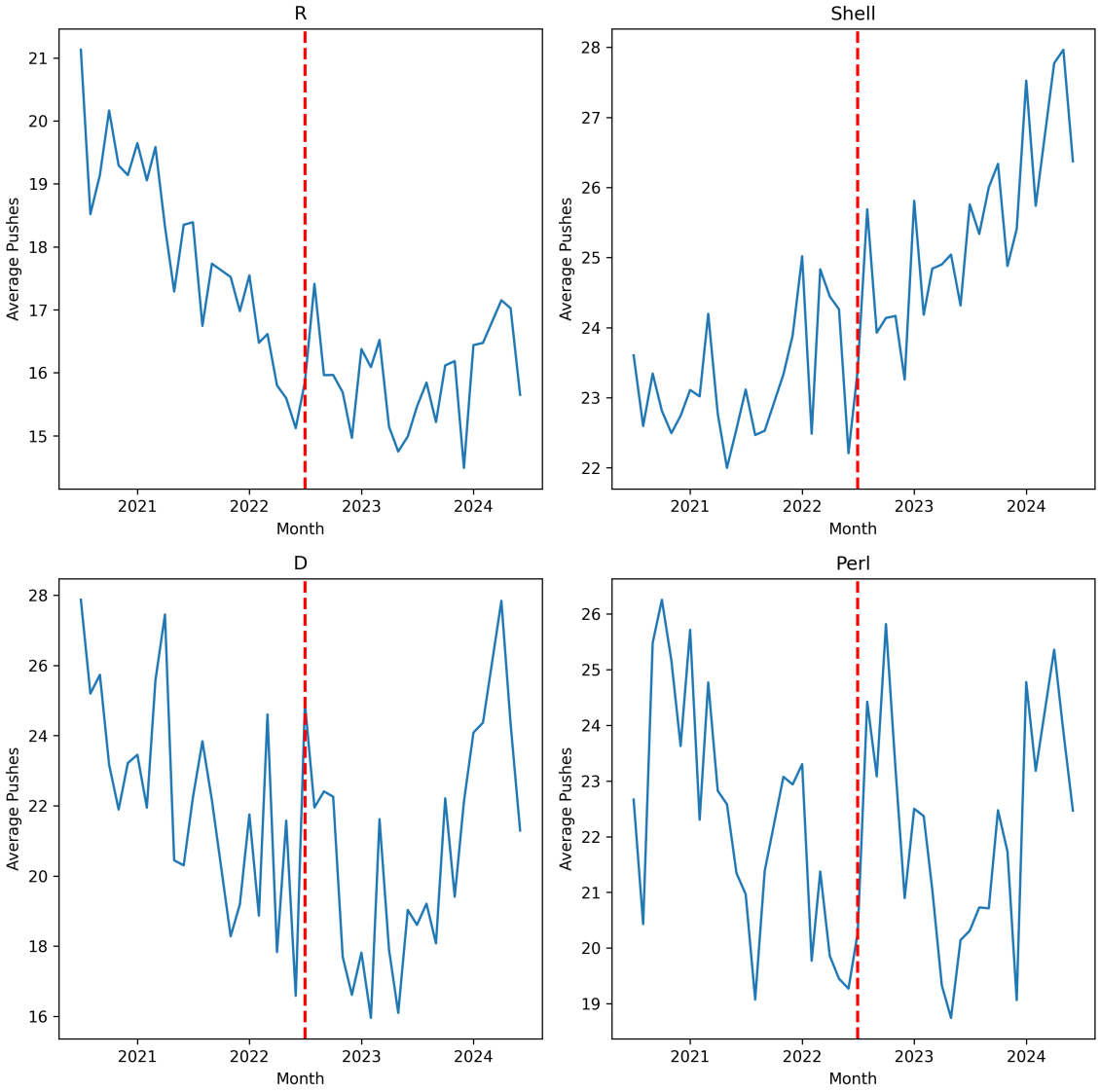
This figure plots . . .

Appendix Figure A3. Pushes per Actor per Month - Treated Languages



This figure plots . . .

Appendix Figure A4. Pushes per Actor per Month - Untreated Languages



This figure plots . . .

B Group Definitions

Language	Avg Pass@1 Rate	Group 1	Group 2	Group 3	Group 4
Python	51.7%	X	X	X	X
JavaScript & TypeScript	49.7%		X	X	X
C++	46.6%			X	X
PHP	46.2%				X
Perl	26.9%				X
D	24.3%			X	X
R	23.8%	X	X	X	X
Shell	21.3%		X	X	X