Generative AI and Market Structure in Open Source Software: Evidence from GitHub Copilot

Michail Batikas*

December 1, 2025

Abstract

A growing body of work documents heterogeneous effects of AI coding assistants on individual developers, but the aggregate consequences for the Open Source Software (OSS) ecosystem remain unclear. We study how GitHub Copilot's launch reshaped OSS market structure. At the market level we find no significant change in the rate of new package entry, but we document an increase in concentration, measured by the Herfindahl–Hirschman Index, and a decrease in inequality, measured by the Gini coefficient, in the Python ecosystem relative to R after Copilot's introduction. We then open the black box with a package-level analysis of 3,213 libraries that "multi-home" in Python and R, exploiting Copilot's staggered rollout as a natural experiment. This difference-in-differences design, implemented with estimators robust to heterogeneous treatment effects, reveals an Adoption Barrier for young, peripheral packages, where AI stimulates community experimentation in forks and watches while significantly reducing downstream dependencies and decoupling experimentation from formal economic adoption. At the same time AI reinforces Core Consolidation, increasing network centrality and adoption for mature packages. Our results suggest that AI may be creating a more vibrant but polarized and potentially fragile OSS ecosystem.

Keywords: Generative AI, Open Source Software, Market Structure, GitHub Copilot, Market Concentration.

JEL Codes: L10, L17, L86, O31, O33

*email: michail.batikas@novasbe.pt, Nova School of Business and Economics

1 Introduction

Generative AI tools are seeing rapid adoption, particularly within the information technology sector and among younger, more educated workers (Hartley et al., 2024), and are already affecting employment dynamics in online labor markets (Hui et al., 2024). In software development specifically, tools like GitHub Copilot are fundamentally altering the field by boosting developer productivity—with some field experiments showing a 26.08% increase in completed tasks (Cui et al., 2025)—and shifting work towards maintenance and refinement (Hoffmann et al., 2024). At the project level, research finds that Copilot adoption increases activity and productivity, but at the cost of significantly higher integration and review times for project maintainers (Song et al., 2025). Studies also show that AI assistance changes the nature of work; developers shift their efforts toward core coding activities and away from project management (Hoffmann et al., 2024), with a notable increase in maintenance-related contributions over new features (Yeverechyahu et al., 2024).

While these individual and project-level effects are becoming clearer, the aggregate impact on the *market structure* of the Open Source Software (OSS) ecosystem remains a critical unknown. Understanding this question is crucial because open-source software serves as critical infrastructure for innovation, and its market structure can influence the rate and direction of technological progress. An ideal OSS marketplace would balance the stability of proven core packages with the creative influx of new ones, allowing superior newcomers to eventually replace outdated incumbents. However, network effects and switching costs often lead to "winner-takes-all" dynamics where incumbents dominate (Arthur, 1989; David, 1985), making such an ideal equilibrium difficult to achieve.

The OSS ecosystem is not a monolith; it is a complex dependency network, a “market for components” where packages compete for developer attention and adoption. In this context, the ’market’ refers to competition among packages *within* a given programming language ecosystem (for instance, two Python libraries offering similar functionality), rather than competition between operating systems or programming languages. This market’s activity

is hosted on public software repositories like GitHub, which serve as transparent hubs for collaboration.

The "components" themselves—the software *packages* (or modules)—are bundles of code that provide some reusable functionality. Developers treat these packages as *inputs* in a production process whose *output* is their own application, script, or analysis. Instead of "reinventing the wheel," they import existing packages much as a manufacturer sources intermediate goods. Language-specific package managers, such as PyPI for Python or CRAN for R, act as the market's supply chain and logistics system: they distribute packages, resolve and install the entire dependency tree, and enforce compatibility constraints.

At a slightly finer level of granularity, a *module* is a unit of code inside a package that performs a cohesive set of operations (e.g., a data loader, a plotting utility). Some modules are relatively self-sustaining; others create value only when combined with additional modules or packages. Crucially, the value of a package is not only what it does in isolation but how easily it can be plugged into the broader system as an input.

The economics literature emphasizes that OSS contributors are often motivated by a mix of intrinsic motives (enjoyment, ideology, learning) and extrinsic rewards, such as reputation, signaling ability to employers, and indirect commercial opportunities (Lerner and Tirole, 2002; Roberts et al., 2006). A package is successful, in this sense, when it is *adopted*—when others build on it, depend on it, and integrate it into their production stack. Downstream dependencies are therefore a natural measure of "economic adoption": more dependencies mean the package has become a key input in others' production functions, increasing its creator's recognition and potential payoffs.

Generative AI tools act as *virtual teammates* in coding (Dell'Acqua et al., 2025), proposing code that frequently imports and uses well-known packages to accomplish tasks. For instance, when a user asks for data manipulation code, an AI assistant is more likely to auto-complete with `pandas` (Python) or `dplyr` (R) than with an obscure new alternative. This behavior can accelerate the adoption of incumbent packages while making it harder for

unfamiliar ones to cross the adoption threshold.

Generative AI's integration therefore creates a central tension: will it proliferate innovation by lowering entry barriers, or consolidate the market by reinforcing incumbents?

On one hand, AI can automate routine coding tasks, lower the skill and effort needed to launch a new project, and increase the set of people who can participate in OSS development. Field experiments suggest large gains for less experienced developers and "casual" contributors, who face the highest initial barriers (Cui et al., 2025; Li et al., 2025). This points to a *Proliferation* force: more experimentation, more packages, and a more dynamic long tail.

On the other hand, AI models are trained on existing public code; they are better at producing code in dominant languages and using popular libraries, creating a "Matthew Effect" where already-popular technologies become even more favored (Gu et al., 2025; Korinek, 2024). Experts working on complex, high-context projects may even slow down when using AI because verifying and correcting suggestions imposes cognitive overhead (Becker et al., 2025). Moreover, recent work shows that post-AI contributions are more maintenance-oriented, concentrating effort into existing repositories rather than spawning entirely new projects (Yeverechyahu et al., 2024; Hoffmann et al., 2024). This suggests a countervailing *Consolidation* force.

We empirically investigate this central tension by leveraging the phased rollout of GitHub Copilot as a natural experiment. We propose and test a contingency model where AI's effect depends on a simple factor: package maturity. The evidence points not to a simple binary, but to a more complex structural transformation. Our central finding is the discovery of an "Adoption Barrier" that decouples community experimentation from formal economic adoption. Our package-level analysis, which forms the core of our causal evidence, reveals this critical decoupling. First, for older, established packages (Core Consolidation), AI acts as a consolidating force, robustly increasing their network centrality and downstream dependencies. Second, for younger, nascent packages (Periphery Experimentation), the impact is

counter-intuitive. While AI sparks a surge in community engagement, experimentation, and contributions, our most robust econometric models (Synthetic DiD) find this activity *fails to translate into formal adoption*. In fact, AI causes a significant *decrease* in downstream dependencies for this group. This suggests AI polarizes the market: it lowers the barrier to experimentation but raises the barrier to adoption, channeling value to the core.

A key methodological strength of this study is the use of a "multi-homing" sample, which conducts "within-package" comparisons between the Python (access to AI tools) and R (no access to AI tools) versions of the same package, providing a highly credible defense against potential confounds like the "ML boom." We make several key contributions. First, we provide some of the first causal evidence of an "Adoption Barrier"—a novel market-level consequence of Generative AI where experimentation is decoupled from adoption. Second, we advance the theoretical debate by proposing and testing a *contingency model* rooted in the *economic theory of modularity*, demonstrating when the forces of proliferation and consolidation dominate. Finally, we raise important questions about the long-term health and resilience of the OSS ecosystem.

Our empirical analysis proceeds in two steps. We first document market-level shifts in concentration and inequality in the Python and R ecosystems, using Herfindahl–Hirschman Index (HHI) and Gini coefficients computed from the dependency network. We then drill down to package-level difference-in-differences designs that exploit within-package comparisons across Python and R versions of the same multi-homing package, allowing us to open the black box behind these aggregate patterns and test the contingency model on core versus peripheral packages. Within the package-level analysis, we estimate our main effects using a Synthetic Difference-in-Differences estimator that is robust to heterogeneous treatment effects, and we verify the robustness of our findings using a conventional two-way fixed effects DiD and the Callaway and Sant'Anna group-time ATT estimator in a dedicated robustness subsection.

We proceed as follows: Section 2 reviews the literature and contrasts two competing

forces, proliferation and consolidation. Section 3 describes the empirical strategy, data, and measures. Section 4 presents market-level evidence on concentration, inequality, and entry. Section 5 then introduces a modularity-based contingency model, the package-level design and outcomes, and reports the package-level DiD results and robustness checks. Section 6 discusses implications, and Section 7 concludes.

2 Literature Review: Proliferation and Consolidation

The scholarly and industry discourse surrounding the impact of Generative AI on the OSS ecosystem is polarized. To understand the mechanisms through which AI may reshape this market, we review two competing forces: one emphasizing proliferation and another emphasizing consolidation.

2.1 Theoretical Force 1: Proliferation

One of two competing theoretical forces is that AI will act as a net positive catalyst for the creation of new OSS packages. The core of this argument rests on AI's ability to lower the technical, economic, and cognitive barriers that have historically constrained software creation, thereby unleashing a new wave of innovation and participation.

A primary mechanism is the democratization of development. AI-powered tools significantly lower the prerequisite technical skill required to create functional software, empowering a larger pool of potential creators. This effect is most pronounced for less-experienced individuals; field experiments show that novice or lower-skilled workers often see the greatest productivity gains from AI assistance, helping them overcome initial skill barriers (Cui et al., 2025). This skill-leveling effect is further supported by research showing that AI can act as a "cybernetic teammate," allowing a single individual to match the creative and problem-solving output of a two-person human team (Dell'Acqua et al., 2025). By making the initial work of architecting and implementing a new project more accessible, AI expands

the population of potential OSS creators.

Another key mechanism is the reduction of opportunity costs. By automating routine and time-consuming tasks, such as writing boilerplate code, generating documentation, and creating tests, AI assistants lower the cognitive and time- related opportunity costs of development. This enables developers to engage in what some have termed "cognitive arbitrage" (Li et al., 2025), freeing up mental resources for more sustained and exploratory activities. Li et al. (2025) provide empirical evidence of this mechanism using the GitHub Copilot X release, finding that developers with AI access increased their commit frequency by 9% and code-related commits by 2%, with a 22% increase in code line additions per commit. Critically, they find these effects are strongest among casual developers, who show a 7% increase in programming language exploration—suggesting AI particularly reduces barriers for those facing higher initial opportunity costs. The reduction in opportunity costs is further evidenced by a 2.2% increase in weekend development activity, indicating AI makes coding more accessible during discretionary time when opportunity costs are traditionally highest.

This individual-level productivity can also translate to the firm level, especially for new ventures. Asam and Heller (2024) find that the release of GitHub Copilot acted as a productivity shock for software-developing startups, significantly reducing their time-to-initial-funding by approximately 20%. This suggests AI lowers the barrier not just to coding, but to viable enterprise creation.

Finally, AI may fuel proliferation by re-tasking expert labor. Research on GitHub Copilot's introduction revealed that its adoption allowed key maintainers to reallocate their time, increasing their share of core coding tasks while reducing their engagement in non-core project management and administrative burdens (Hoffmann et al., 2024). By offloading administrative duties, AI could free up expert developers to engage in more innovative, "zero-to-one" creation.

2.2 Theoretical Force 2: Consolidation

The opposing viewpoint is that AI's integration will ultimately lead to a consolidation of effort and influence within a smaller set of established packages, stifling the creation of new, independent alternatives. This argument is built on evidence of productivity paradoxes, powerful ecosystem-level feedback loops, a fundamental shift in contribution patterns, and new governance burdens.

A compelling body of research reveals a *productivity paradox* for experts, where AI tools can hinder more than they help in complex, high-context environments. A rigorous randomized controlled trial involving experienced open- source developers working on their own mature projects found that developers using AI tools actually took 19% longer to complete their tasks (Becker et al., 2025). This counterintuitive slowdown stems from the substantial cognitive load required to manage the AI's output. Developers report spending significant time simply verifying and debugging AI-generated suggestions, which can be subtly flawed, insecure, or misaligned with a project's implicit architectural principles (Zhang et al., 2025). This "validative overload" can easily negate any time saved during initial code generation.

A second, powerful mechanism is the *data feedback loop* and the "Matthew Effect." Generative AI models create a "rich-get-richer" dynamic that reinforces the dominance of established ecosystems. At a macro level, the market for foundation models itself is prone to concentration due to immense scaling costs and data requirements, creating a few powerful "gatekeepers" of intelligence (Korinek, 2024). At a micro level, the content of the training data itself biases model output. The models are trained on vast corpora of public code, which are heavily skewed toward the most popular languages and libraries. This training bias means AI assistants are inherently more proficient at generating code for dominant technologies. Gu et al. (2025) provide direct empirical evidence of this phenomenon, terming it a "Matthew Effect": the more popular a programming language or framework, the higher the success rate of LLM-generated code. This effect is not limited to code; research on LLM training data shows that model performance is significantly better on content (like books)

that was included in pirated training datasets, an effect that is ironically strongest for less popular books, suggesting AI training can amplify obscure content it was fed, at the expense of content it never saw (Jia and Nagaraj, 2025). This overall effect risks creating technological lock-in, steering developers toward a narrow set of well-supported tools and stifling innovation in the long tail of the ecosystem. This guidance mechanism, inherent in the AI's suggestions, may create a scenario where AI *simultaneously* lowers the barrier to *creating* novel packages while *raising* the barrier to their *adoption* by steering users to established incumbents.

A third mechanism is a *shift from origination to iteration*. AI appears to be fundamentally altering the nature of developer contributions. Research leveraging the staggered rollout of GitHub Copilot found a 5.9% to 6.5increase in project-level activity (Song et al., 2025). However, Yeverechyahu et al. (2024) found this increased activity predominantly comprised maintenance-related contributions rather than new code development. If AI primarily channels developer effort toward improving existing projects rather than inspiring new ones, the logical outcome is a consolidation of activity within popular, established repositories. This is compounded by the finding that while AI increases the volume of code contributions, it also increases the coordination time for code integration and review, placing a heavier burden on project maintainers (Song et al., 2025).

This leads to the fourth mechanism: an *increased burden on core maintainers*. The influx of AI-assisted contributions, primarily from less- experienced developers, creates a new maintenance bottleneck. Research finds that code written after AI adoption requires more rework, and this burden falls squarely on the more experienced core developers who must review and fix these contributions (Hoffmann et al., 2024). This dynamic exacerbates the well- documented problem of maintainer burnout, a critical threat to the sustainability of the OSS ecosystem.

We summarize the core arguments and mechanisms derived from this literature in Table 1.

Table 1: Summary of Literature on AI's Impact on Productivity and Innovation

Authors	Result
Studies with Positive Findings	
Peng et al., 2023	Positive: Significant increase in task completion speed (up to 55% faster) and volume, especially for less-experienced developers on well-defined tasks.
Dell'Acqua et al., 2025	Positive: Augments individual creativity and problem-solving, enabling individuals to match small team performance and democratizing expertise.
Studies with Negative Findings	
Becker et al., 2025	Negative: Experienced developers on complex, familiar open-source projects take 19% longer with AI assistance due to verification overhead.
Gu et al., 2025	Negative: Reinforces dominance of popular languages/frameworks through a "Matthew Effect," threatening ecosystem diversity.
Zhang et al., 2025	Negative: Imposes significant cognitive load ("validative overload"), requiring considerable time verifying and debugging flawed AI suggestions.
Studies with Heterogeneous Findings	
Brynjolfsson et al., 2025	Heterogeneous: Positive for overall productivity (+14%) and novices (+34%). Negative for the most skilled workers (slight quality decline).
Hoffmann et al., 2024	Heterogeneous: Positive for less-experienced (peripheral) developers. Negative in increasing code requiring rework, shifting maintenance burden to experienced (core) developers.
Song et al., 2025	Heterogeneous: Positive in increasing project-level contributions (+5.9%). Negative in increasing coordination time (+8%), creating productivity tradeoff.
Yeverechyahu et al., 2024	Heterogeneous: Positive in boosting contribution volume (commits +33%). Negative in concentrating increase in iterative maintenance (+24%) vs. novel development (+13%).
Li et al., 2025	Heterogeneous: Positive for casual developers (7% increase in language exploration, 2.2% in weekend development). Effects stronger for less-experienced developers. Finds medium-term career benefits.

3 Empirical Strategy

To causally identify the impact of Generative AI on the OSS market structure, this study leverages the introduction of GitHub Copilot as a natural experiment. The identification strategy hinges on a crucial feature of the tool's initial release: selective language support, where certain programming languages received support while others did not. This variation was not an arbitrary design choice but a direct consequence of the underlying training data

used to build the AI model and the technical capabilities of the initial release. This section details the technological shock, justifies the selection of treatment and control groups based on this selective rollout, and outlines the data used in our analysis.

3.1 GitHub Copilot

GitHub Copilot, an AI-powered pair-programming tool, was developed by GitHub in collaboration with OpenAI to function as an assistive technology that autocompletes code directly within a developer's integrated development environment (IDE) (Gershgorn, 2021). The tool was first announced and released as a technical preview on June 29, 2021, for the Visual Studio Code environment (Friedman, 2021). The official launch, which serves as the "treatment event" for this study's DiD analysis, occurred in October 2021. This timing is anchored by its broad release as a plugin on the widely used JetBrains marketplace on October 29, 2021 (*GitHub Copilot*, 2024).

The initial version of GitHub Copilot was powered by OpenAI Codex, a large language model explicitly derived from the GPT-3 family and fine-tuned for code generation (Zaremba, 2021). The efficacy of such a model is a direct function of the quality and quantity of its training data for a specific domain. Codex was trained on a massive corpus of publicly available source code, with the majority sourced from public repositories on GitHub. Critically, the composition of this training data was heavily skewed toward certain languages. The dataset included a filtered corpus of 159 gigabytes of Python code, sourced from over 54 million public GitHub repositories (Chen et al., 2021). While the training set included numerous other languages (e.g., JavaScript, Go, Ruby), the sheer volume of Python data gave the model a significant, inherent performance advantage in that language from its inception.

This data composition had direct implications for language support. Upon launch, GitHub and its partners explicitly stated that the technical preview worked "especially well for Python, JavaScript, TypeScript, Ruby, and Go" (Friedman, 2021). Languages not included in this list, such as R, were simply not supported by the initial release. Importantly,

this absence of R support stemmed from technical training-data limitations rather than any strategic market consideration, underscoring the exogeneity of our shock for the R ecosystem.

3.2 Identification Strategy

Our investigation contrasts community contributions to open-source packages across Python and R, two prominent data-oriented programming languages with active developer communities. Both language ecosystems are supported by a wealth of libraries and frameworks (i.e., packages), such as Python's NumPy and R's ggplot2. The open-source communities for these languages are hosted on GitHub, a platform central to the ecosystem of open-source projects with over 100 million software developers contributing across more than 500 programming languages. On GitHub, communities organize around 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.

The natural experiment that our investigation exploits is the launch of GitHub Copilot in October 2021. Upon its official launch, GitHub Copilot selectively supported certain programming languages, including Python, JavaScript, TypeScript, Ruby, and Go, while not supporting others such as R and Haskell. This was not a matter of differential effectiveness but of explicit non-support. By the end of 2022, ChatGPT was launched, and by 2023 GitHub Copilot eventually broadened its support to include previously unsupported languages like R. At least until the ChatGPT launch, languages like R did not have mainstream LLM coding assistance. We thus focus our analysis on this window (pre-2023) where Python had AI assistance and R effectively did not. This also implies that our estimates likely represent a lower bound of AI's impact, as any informal adoption of ChatGPT by R developers after late 2022 (or eventual Copilot support for R in 2023) would only narrow the Python–R gap.

In our empirical design, Python packages serve as the treatment group, receiving access to

GitHub Copilot's productivity-enhancing capabilities starting in October 2021. R packages serve as the control group, as R was explicitly not supported by Copilot during the initial rollout period through 2022. The treatment period thus spans from October 2021 through the end of 2022, during which Python developers had access to Copilot while R developers did not have access to any mainstream LLM coding assistance. This creates a clean comparison: the effect of LLM-assisted development (Python) versus traditional development without such assistance (R).

However, we must acknowledge an important identification challenge. During the pre-treatment period, the rise of machine learning frameworks such as TensorFlow and PyTorch created an exogenous demand shock for Python, which has become the lingua franca of machine learning and data science. R, a more specialized language for statistical computing, did not experience the same demand shock. This divergence in demand trends represents a classic confounding variable that could threaten the parallel trends assumption underlying our difference-in- differences design. Differential pre-existing trends between Python and R ecosystems could bias our estimates of Copilot's causal impact.

Our aggregate (market-level) analysis therefore incorporates language-specific linear time trends and is interpreted as suggestive context rather than our primary source of causal identification. Our main causal evidence will come from a package-level analysis that exploits multi-homing packages and more flexible DiD estimators, presented in Section 5.

3.3 Data and Overall Sample

A comprehensive dataset was constructed by integrating information from multiple sources. The primary sources include PyPI (Python Package Index) and CRAN (Comprehensive R Archive Network) for the universe of packages and metadata, GitHub for development activity (forks, watches, contributors), and deps.dev/libraries.io for inter-package dependency data. This dataset, which covers the universe of Python and R languages, spans from February 1998 to August 2024. It includes a total of 526,412 packages, consisting of 498,715

Python packages and 27,967 R packages. This large disparity in ecosystem size suggests that Python's ecosystem is much broader, while R's is more specialized; indeed, R's smaller scale may indicate higher entry barriers or a more niche focus.

The aggregate analysis in Section 4 uses this full universe of packages to compute market-level indicators (HHI, Gini, entry rates) for each language-month. The package-level analysis in Section 5 will focus on a more tightly defined subset of multi-homing packages and introduce additional design details and outcome measures at that stage.

4 Market-Level Evidence

We begin by examining how GitHub Copilot's introduction relates to market-level patterns of concentration, inequality, and entry in the Python and R ecosystems. This aggregate evidence provides a high-level picture of structural change that helps motivate, and is later interpreted through, the package-level mechanisms in Section 5.

4.1 Aggregate Model and Limitations

This aggregate analysis faces a significant challenge: the Python and R ecosystems exhibited different pre-treatment trends in market concentration. A formal test confirms these trends differ significantly ($F = 197.75$, $p < 0.0001$). This violation of the parallel trends assumption means a standard DiD estimator will yield biased results. Following Autor (2003) and Wolfers (2006), we account for these differential trends by including group-specific linear trends in our two-way fixed effects specification:

$$Y_{pt} = \alpha + \beta \cdot (Python_p \times Copilot_t) + \gamma_p \cdot t + \delta_p + \mu_t + \varepsilon_{pt}, \tag{1}$$

where Y_{pt} is the market-level metric for language p in month t. $Python_p$ is a time-invariant dummy for Python language, and $Copilot_t$ is a dummy for months after October 2021. $\gamma_p \cdot t$

represents group-specific (language-specific) linear time trends. δ_p are language fixed effects, and μ_t are calendar-month fixed effects.

This "fix," however, relies on the strong and untestable assumption that the sole reason for the divergence was a simple linear trend and that both ecosystems would have continued to diverge at that exact rate. It is for this reason that we ultimately treat the package-level analysis in Section 5 as our primary source of causal identification, and use the aggregate evidence here as descriptive and suggestive context.

The market-level metrics are:

- **Herfindahl-Hirschman Index (HHI):** A measure of market concentration, calculated as the sum of squared market shares of downstream dependencies.
- **Gini Coefficient:** A measure of inequality in the distribution of downstream dependencies.
- **New Package Entry Rate:** The number of new packages created each month.

To visualize these dynamics, Figure 1 plots the monthly HHI for Python and R, and Figure 2 plots the corresponding Gini coefficients. Both figures show the pre-treatment divergence and the shift in trend around Copilot's introduction. On the other hand, the growth of both ecosystems seems to be identical, as seen in Figure 2.

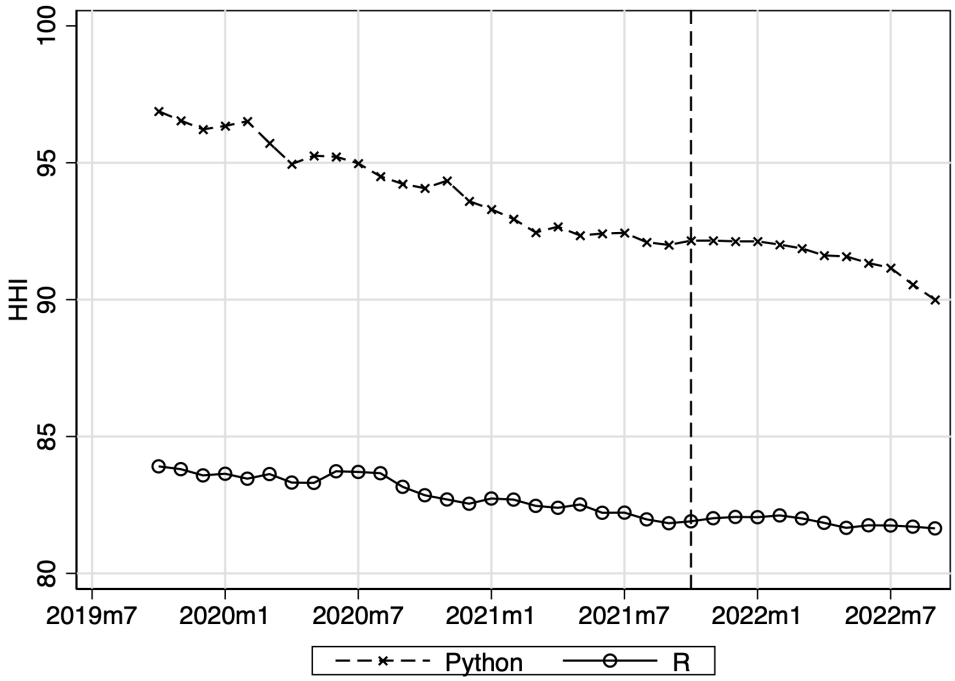


Figure 1: Monthly HHI for Python and R (Dependency-Based Market Shares)

Notes: Figure shows the evolution of the Herfindahl-Hirschman Index (HHI) for Python and R based on downstream dependencies. The vertical line marks October 2021, the Copilot launch.

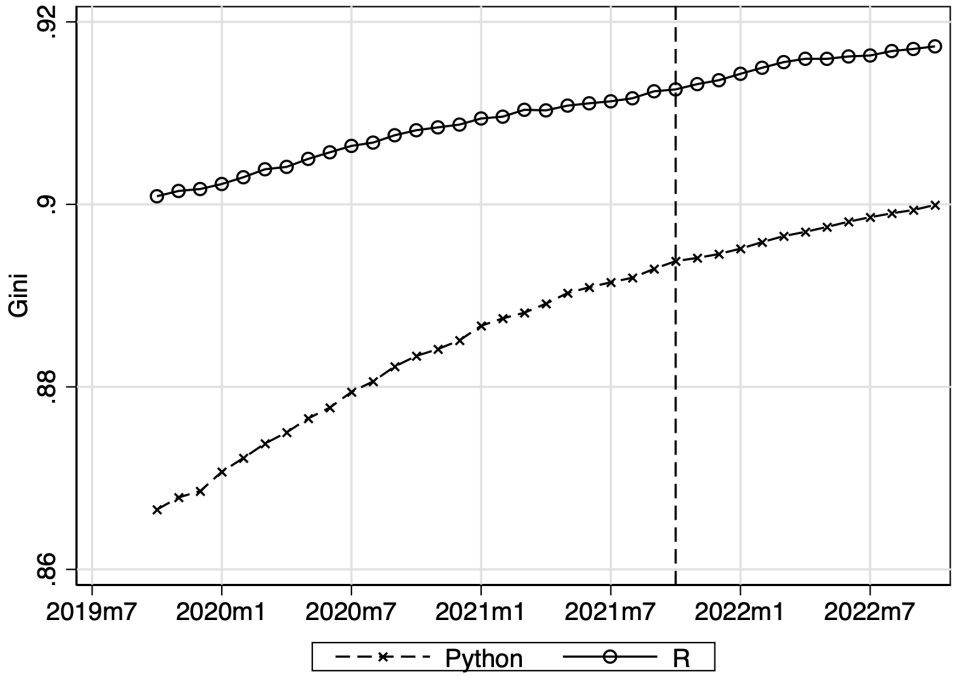


Figure 2: Monthly Gini Coefficients for Python and R (Downstream Dependencies)

Notes: Figure shows the evolution of the Gini coefficient for Python and R based on the distribution of downstream dependencies across packages. The vertical line marks October 2021.

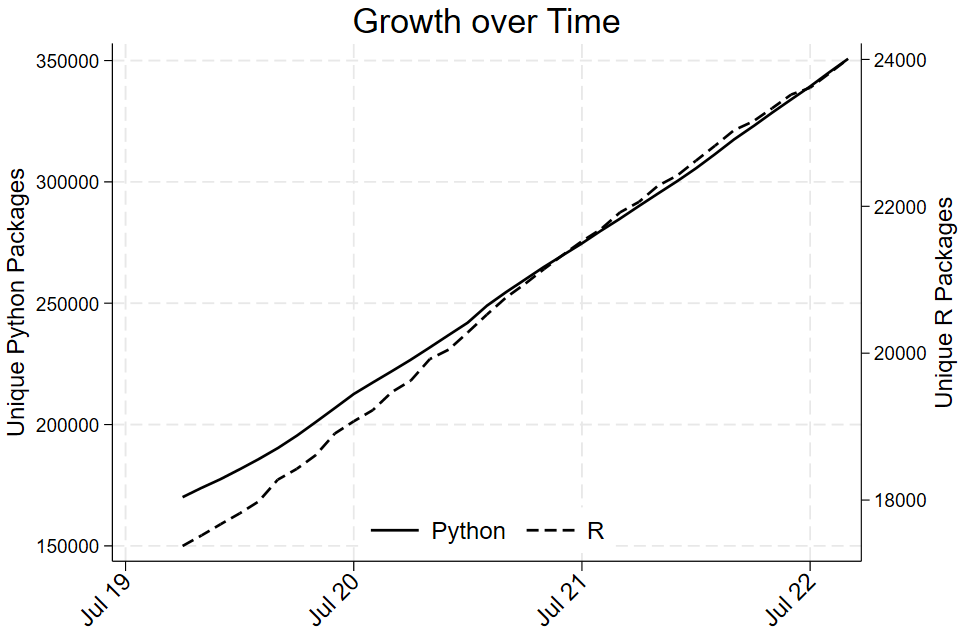


Figure 3: Cumulative number of unique packages for Python and R

Notes: Figure shows the evolution of the ecosystem for Python and R based on the total number of unique packages.

4.2 Structural Polarization

The results of the aggregate DiD specification, presented in Table 2, show a pattern consistent with our later package-level findings. We find a significant increase in market concentration (HHI) (Coeff. $= 0.925$, $p < 0.01$) and a simultaneous, significant decrease in inequality (Gini) (Coeff. $= -0.004$, $p < 0.01$). Finally, we find no statistically significant change in the (log) rate of new package entry (Coeff. $= 0.126$, $p > 0.1$).

Table 2: Aggregate Market-Level Effects

Dependent Variable	Coeff. (Std. Err.)
Market Concentration (HHI)	0.925^{***} (0.193)
Gini Coefficient	-0.004^{***} (0.001)
New Package Entry Rate (log)	0.126 (0.120)

Notes: Table displays the coefficient on the interaction term ($Python \times Copilot$) from DiD regressions estimated at the language-month level. Standard errors are clustered at the language level. $^{*}p < 0.1$; $^{**}p < 0.05$; $^{***}p < 0.01$. Model includes language and time fixed effects, as well as group-specific linear trends.

This combination of rising concentration but falling inequality is informative. Visually, Figure 1 shows that the HHI for Python increases relative to R after Copilot, indicating that a larger share of dependency weight is accruing to the top packages. At the same time, Figure **??** shows a slight decline in the Gini coefficient, implying that once we move beyond the very top, the distribution of dependencies across the remaining packages becomes somewhat more equal.

One interpretation consistent with our package-level results is that AI is intensifying the dominance of a small core of "superstar" packages while flattening the rest of the distribution among peripheral ones. Entry rates do not change significantly, suggesting that Copilot did not meaningfully increase the number of new packages being published in the short run. Instead, it appears to have shifted how activity and adoption are distributed among existing packages.

4.3 Robustness Check

To validate this aggregate specification, we conduct a falsification test by shifting the treatment date one year prior (a "false" treatment date) and excluding the true post-treatment period. The results, presented in Table 3, show a complete absence of statistically significant effects. This lack of any significant findings during a period where no treatment actually occurred provides confidence that the effects reported in Table 2, while suggestive, are attributable to the introduction of Copilot rather than to spurious, pre-existing differential trends not captured by the linear trend model.

Table 3: Aggregate Market-Level Effects - Falsification Test

Dependent Variable	Coeff. (Std. Err.)
Market Concentration (HHI)	0.336 (0.266)
Gini Coefficient	0.000 (0.001)
New Package Entry Rate (log)	−0.031 (0.247)

Notes: Table displays the coefficient on the interaction term ($Python \times False\ Copilot$) from DiD regressions estimated at the language-month level. Standard errors are clustered at the language level. $^{*}p < 0.1$; $^{**}p < 0.05$; $^{***}p < 0.01$. Model includes language and time fixed effects and group-specific linear trends.

Taken together, the aggregate evidence suggests that Copilot coincides with a structural polarization of the Python ecosystem: greater concentration at the top, a somewhat more even distribution among the rest, and no clear change in entry. In the next section, we show that these macro patterns are underpinned by distinct micro-level responses of core versus peripheral packages.

5 Package-Level Mechanisms

We now turn to the package-level analysis, which opens the black box behind the aggregate patterns documented in Section 4. In this section, we first introduce the modular architecture of OSS and our contingency model and hypotheses, and then detail the package-level sample, empirical specification, and outcome measures before presenting the main results

and robustness checks.

5.1 Modularity and the Architecture of OSS

The collaborative, decentralized nature of OSS development is made possible by a core architectural principle: modularity. Modularity involves organizing a complex system into smaller, self-contained, and independent modules, each with a specific function and a clear, stable interface (vFunction, 2024; Hester and Ellis, 2016). The objective is to manage complexity by ensuring that elements within a module are strongly interconnected while connections *between* modules are weak and well-defined, a concept known as "loose coupling" and "high cohesion" (MacCormack et al., 2008).

The seminal work on the economics of modularity, Baldwin and Clark (2000), argues that it is not merely a technical convenience but a powerful economic force. A modular architecture partitions a complex design into "visible" design rules—the public interfaces and standards that govern how modules interact—and "hidden" modules, which are the private, internal workings of each component (Ethiraj and et al., 2022; Baldwin, 2007). This separation creates valuable economic "options": the ability to experiment with, improve, or completely replace individual hidden modules without destabilizing the entire system, so long as the visible design rules are obeyed. This structure fosters parallel innovation and dramatically accelerates industry evolution (Leading Through Staff, 2023; Baldwin and Clark, 2000).

The modular architecture of the software itself allows a distributed community of contributors to focus on specific components without needing to understand the entire system, facilitating a global division of labor and enabling parallel development (Narduzzo and Rossi, 2005). In the OSS context, the "visible design rules" are the public APIs of a library, the documented contribution processes, and the established coding standards. These are not just technical specifications but are socio-technical contracts that enable trust and permissionless innovation. A developer can confidently build their own "module" (a new package)

that depends on a core library, trusting that its API will remain stable. The success of the modular OSS ecosystem thus depends on the stability and clarity of these contracts.

5.2 A Contingency Model and Hypotheses

The literature review in Section 2 reveals competing forces. We argue these forces are not mutually exclusive but can be unified and predicted by applying the economic theory of modularity (Section 5.1) to the mechanisms of Generative AI (Sections 2.1 and 2.2). As established in Section 5.1, the OSS ecosystem's value derives from "parallel innovation" based on a modular architecture: "visible design rules" (the stable APIs of core, incumbent packages) and "hidden modules" (new, experimental packages) (Baldwin and Clark, 2000). AI's mechanisms act upon these components to produce heterogeneous outcomes.

First, the consolidation force (Section 2.2) acts by strengthening the dominant "visible design rules." AI models are trained on vast corpora of public code, making them experts at using popular, well- documented packages (the "Matthew Effect"). AI's training bias acts as a mechanism to *strengthen the dominance of 'visible design rules.'* It lowers the cognitive cost for developers to write code that conforms to these dominant APIs, as the AI suggests this conforming code automatically. This channels dependencies toward the core incumbents.

Hypothesis 1 (Core Consolidation) *The introduction of Generative AI will lead to:*

Hypothesis 1a *A significant increase in network centrality for mature, established packages.*

Hypothesis 1b *A significant increase in formal adoption (downstream dependencies) for mature, established packages.*

Second, the proliferation force (Section 2.1) acts by lowering the cost of creating new "hidden modules." AI's barrier-lowering capability acts as a mechanism to *reduce the cost of creating new 'hidden modules.'* It automates boilerplate code and lowers skill barriers, making it easier for developers to experiment with new ideas (i.e., create new packages, fork existing ones).

Hypothesis 2 (Periphery Experimentation) *The introduction of Generative AI will lead to:*

Hypothesis 2a *A significant increase in community engagement (e.g., forks, watches, contributors) for nascent, peripheral packages.*

Hypothesis 2b *The introduction of Generative AI will lead to a significant decrease in formal adoption (downstream dependencies) for nascent, peripheral packages.*

Herein lies the central theoretical tension. We posit that AI simultaneously lowers the cost of creation (H2a) while increasing the relative cost of trusting and adopting a new, unproven 'hidden module.' Because the AI, biased by its training data, will almost always suggest the incumbent "good enough" module (H1), a developer must exert more cognitive effort to ignore the AI's suggestion and manually search for, vet, and adopt a nascent alternative. This increases the relative search and verification cost, an effect related to the "validative overload" discussed in the literature (Becker et al., 2025). This leads to our "Adoption Barrier" hypothesis, which posits a decoupling of experimentation (H2a) from adoption.

5.3 Package-Level Design and Sample

To understand the mechanisms driving market-level effects and test our contingency model, we conduct a complementary package-level analysis. We construct a balanced sample of multi-homing packages—packages that maintain versions in both Python and R ecosystems. This "multi-homing" sample is our primary defense against the "ML boom" confound identified in Section 3.2.

Multihoming packages are software libraries that provide equivalent functionality across multiple programming languages, with separate implementations for Python and R. Examples include data manipulation libraries (e.g., `dplyr` in R and `pandas` in Python), visualization tools, and statistical packages that serve the same purpose across both ecosystems.

The use of multihoming packages offers several critical advantages for identification. First, these packages face similar demand conditions across both languages, as they target overlapping user bases with similar functionality needs. A data scientist requiring data manipulation capabilities represents potential demand for both the Python and R versions of such a package. This helps mitigate concerns about differential demand shocks, as the underlying user needs drive demand for both implementations. Second, multihoming packages are more directly comparable in their market positioning and competitive environment. They occupy similar niches within their respective ecosystems, reducing concerns about compositional differences between Python and R packages. Third, focusing on multihoming packages helps address concerns about unobserved heterogeneity. Any package-specific factors that might drive trends in contributions (e.g., the general appeal of the functionality, the size of the potential user base, or the maturity of the problem domain) affect both the Python and R versions, making these factors less likely to confound our estimates.

By restricting our sample to multihoming packages, we effectively compare Python and R implementations of the same underlying software concepts. The key identifying assumption becomes more plausible: conditional on package-level fixed effects and any differential pre-trends, the introduction of Copilot represents an exogenous shock to productivity that differentially affected Python implementations (which received support) relative to R implementations (which did not). This represents a stronger quasi-experimental design than a simple comparison of all Python versus all R packages, as it controls for demand-side factors by construction.

Multihoming packages are identified by matching package names across PyPI and CRAN, yielding 3,213 packages with presence in both languages. This sample is particularly valuable for two reasons. First, by comparing the Python version of a package to the R version of the exact same package, we create a powerful quasi-experimental design. Second, multi-homing packages represent strategically important software that serves both communities, making their response to Copilot especially informative about cross-language dynamics.

The package-level dataset uses the same temporal coverage as the language-level analysis, spanning October 2019 to September 2022, providing 24 months of pre- treatment data and 12 months of post-treatment data. The unit of analysis is the package-language-month, with each of the 3,213 multi-homing packages observed in both Python and R versions over 36 months, resulting in 231,336 package- language-month observations.

To investigate whether Copilot's effects vary by package maturity, we construct an age measure based on each package's tenure as of September 2021—the month immediately preceding Copilot's introduction. Package age is defined as the number of months elapsed since the package's first release on its respective platform (PyPI or CRAN) through September 2021. This snapshot approach ensures that the age variable is pre-determined and not influenced by post-treatment dynamics. We partition the multi-homing sample into approximately equal-sized quartiles based on package age: Q1 (youngest packages), Q2, Q3, and Q4 (oldest packages). This quartile-based approach allows us to test our contingency model (Hypotheses 1 and 2) directly. In our sample, older packages (Q4) indeed have considerably more adoption and engagement than younger ones (Q1) prior to Copilot's introduction. For instance, by September 2021, a typical Q4 package had on the order of tens of downstream dependents and a substantial number of forks and stars, whereas a typical Q1 package had little to no adoption (zero or very few dependents) and minimal community activity. This descriptive contrast highlights the core-periphery distinction in the OSS ecosystem even before the AI shock.

5.4 Package-Level Empirical Specification

Our package-level analysis employs a two-way fixed effects difference-in- differences specification, which serves as a baseline model:

$$Y_{ipt} = \alpha + \beta \cdot (Python_{ip} \times Copilot_t) + \delta_{ip} + \mu_t + \varepsilon_{ipt}, \tag{2}$$

where Y_{ipt} is the outcome variable for package i in language p at time t. $Python_{ip}$ is a dummy variable equal to one for the Python version of package i and zero for its R version. $Copilot_t$ is a post-treatment indicator for months after October 2021. δ_{ip} are package-language fixed effects that control for time- invariant differences between the Python and R versions of each package. μ_t are calendar-month fixed effects that absorb common temporal shocks affecting all packages. Standard errors are clustered at the package level to account for serial correlation within packages over time.

The coefficient of interest, β, identifies the causal effect of Copilot on the Python version of multi-homing packages relative to their R counterparts. We estimate Equation 2 separately for each age quartile to assess heterogeneous treatment effects across package maturity levels. Given the importance of treatment-effect heterogeneity in our setting, we also employ Callaway and Sant'Anna (C&S) and Synthetic DiD estimators that are robust to such heterogeneity; these are the focus of our main results.

5.5 Package-Level Outcome Measures

We create several package-level dependent variables to capture different dimensions of package influence, adoption, and community engagement. All metrics are calculated at the monthly level for each package:

- **Eigenvector Centrality:** A measure of a package's influence within the dependency network, calculated monthly across the entire universe of packages in each language ecosystem. This variable has been calculated using the monthly data of all Python and R packages. A package has high eigenvector centrality if it is connected to other highly central packages. Formally, eigenvector centrality is defined as:

$$x_{ipt} = \frac{1}{\lambda_t} \sum_{j=1}^{M_{pt}} A_{jipt} x_{jpt}$$

where x_{ipt} is the centrality score for package i in language p at month t, A_{jipt} is

an element of the adjacency matrix representing the dependency relationship (where $A_{jipt} = 1$ if package j depends on package i, and 0 otherwise), λ_t is the largest eigenvalue of the adjacency matrix, and M_{pt} is the total number of packages in the entire ecosystem (all packages in language p at month t, not just those in the analytical sample). This metric captures a package's status as a core component of the ecosystem.

- **Downstream Dependencies:** The number of other packages that list a given package as a dependency. This is a direct measure of a package's adoption, and importance within the ecosystem.

- **Upstream Dependencies:** The number of other packages that a given package relies on. This metric reflects the complexity and modularity of a package.

- **Forks:** The number of times a package's GitHub repository has been copied (forked) by users. This is extracted directly from GitHub's repository metadata and reflects community interest, experimentation, and potential for derivative work. A fork represents a user creating their own copy of the repository to potentially modify or build upon.

- **Watches (Stars):** The number of GitHub users who have starred a repository, measured monthly. This metric is extracted from GitHub's repository metadata and reflects a package's visibility, popularity, and community attention.

- **Contributors:** The number of unique developers who have contributed code to the package's repository. This metric captures the breadth of participation and collaborative input a package receives.

5.6 Main Results: Core Consolidation and the Adoption Barrier

Whereas the market-level evidence in Section 4 reveals a broad pattern of increasing concentration and structural polarization, the package-level regressions allow us to unpack the

mechanisms and heterogeneous treatment effects across package maturity.

Our primary package-level analysis tests our contingency model, investigating how Copilot's introduction affected packages differentially based on their maturity. We present results for key outcome variables covering both adoption and community engagement, with a focus on differences between older (core) and younger (peripheral) packages.

Table 4 presents our preferred specification, the Synthetic DiD estimator, which is robust to heterogeneous treatment effects. The Synthetic DiD model finds a strong positive effect on *Eigenvector Centrality* for older (Q4) packages (Coeff. $= 0.0073$, $p < 0.01$), supporting Hypothesis 1a, and a positive effect on *Downstream Dependencies* for older packages (Coeff. $= 0.0396$, $p < 0.01$), supporting Hypothesis 1b. This provides clear evidence for the 'Core Consolidation' hypothesis: AI reinforces the dominance of core, established packages, leading to a "Matthew Effect" where the rich get richer. The community engagement metrics (*Forks*, *Watches*) for this group were largely unaffected, indicating AI's impact was not on attracting new experimenters but on converting existing interest into greater adoption and centrality for these incumbent packages.

For the youngest (Q1) packages, the story is the opposite. The Synthetic DiD estimator (Table 4) finds a large, statistically significant *negative* effect on *Downstream Dependencies* (Coeff. $= -0.2514$, $p < 0.01$). That is, new packages in the Python ecosystem saw a decrease in formal adoption (being depended upon by other packages) relative to their R counterparts. This result supports Hypothesis 2b, the "Adoption Barrier." Interestingly, we observe a significant *increase* in community engagement for these same packages: *Forks* and *Watches* rise significantly for Q1 Python packages (e.g., Watches Coeff. $= 0.263$, $p < 0.05$ in the Synthetic DiD), consistent with Hypothesis 2a that AI enables more experimentation and tinkering with new projects. However, this experimentation surge does not translate into dependency gains—in fact, dependencies decline. This divergence between engagement and adoption for new packages is precisely the AI-induced decoupling that constitutes the Adoption Barrier.

The middle quartiles (Q2 and Q3) generally show intermediate effects. We see slight increases or flat effects for centrality and adoption metrics (most not significant or smaller in magnitude), suggesting that moderately established packages were not strongly affected in either direction. This pattern is intuitive: packages that are neither very new nor top-tier incumbents might neither benefit greatly from AI's reinforcement of popularity nor suffer from being overlooked.

Finally, to corroborate the dynamic nature of these effects, Appendix presents event-study plots for representative outcomes. The parallel trends hold in the pre-treatment period for all outcomes, and the divergence after Copilot launch (October 2021) is visually apparent: older packages trend upward relative to control, while the youngest packages trend downward in adoption despite upticks in engagement signals.

Table 4: Difference-in-Differences Results (Synthetic DiD): Heterogeneity by Package Age

	Youngest Packages (Q1)	Oldest Packages (Q4)
Dependent Variable	Coeff. (Std. Err.)	Coeff. (Std. Err.)
Eigenvector Centrality	0.00000^{***} (0.00000)	0.00728^{***} (0.00010)
Upstream Dependencies	0.02828^{***} (0.00568)	0.00799^{***} (0.00276)
Downstream Dependencies	-0.25141^{***} (0.06624)	0.03963^{***} (0.00925)
Forks	0.16633^{***} (0.02017)	0.04290^{**} (0.01691)
Watches	0.14066^{***} (0.02225)	0.00394 (0.02050)
Contributors	0.10680^{***} (0.03643)	0.02914 (0.02526)

Notes: Table displays the Average Treatment Effect on the Treated (ATT) from Synthetic DiD regressions. The full regression outputs for each variable and estimator are available in Appendix. Standard errors clustered at the package level are in parentheses. $^{*}p < 0.1$; $^{**}p < 0.05$; $^{***}p < 0.01$.

The evidence from our preferred specification paints a consistent picture: AI introduced via Copilot has been a force for consolidation at the core and a barrier to adoption at the periphery. Notably, simpler models that do not account for treatment effect heterogeneity (e.g., a naive TWFE on the full sample) would miss or misstate this nuance. Indeed, when we estimate a conventional TWFE DiD pooling all packages, we find a small positive average effect on new downstream dependencies—masking the large positive effect for cores and the negative effect for new entrants that cancel out in aggregate. This underscores

the importance of examining heterogeneity in contexts where new technology shocks have distributional consequences.

These micro-level patterns align with the macro-level shifts documented in Section 4: the strengthening of the core manifests as higher HHI, while the inhibited adoption of peripheral packages is consistent with a flatter, but more fragile, long tail.

5.7 Robustness: Alternative DiD Estimators

To assess the robustness of our findings, we re-estimate the package-level effects using two additional difference-in-differences estimators: a conventional two-way fixed effects (TWFE) specification (Table 5) and the Callaway and Sant'Anna (C&S) (Table 6) group-time ATT estimator. In Appendix we report the full set of estimates for all three estimators (TWFE, C&S, and Synthetic DiD) across age quartiles and outcome variables.

Table 5: Difference-in-Differences Results (TWFE): Heterogeneity by Package Age

	Youngest Packages (Q1)	Oldest Packages (Q4)
Dependent Variable	Coeff. (Std. Err.)	Coeff. (Std. Err.)
Eigenvector Centrality	−0.00160 (0.00430)	0.00984** (0.00481)
Upstream Dependencies	0.07393*** (0.01848)	0.02280** (0.01135)
Downstream Dependencies	0.06560*** (0.01554)	0.11578*** (0.01491)
Forks	0.01919** (0.00769)	−0.00003 (0.00621)
Watches	0.04257*** (0.01406)	−0.01091 (0.00981)
Contributors	0.03986** (0.01595)	0.00191 (0.01005)

Notes: Table displays the coefficient on the interaction term from TWFE DiD regressions, representing the Average Treatment Effect on the Treated (ATT). The full regression outputs for each variable and estimator are available in Appendix. Standard errors clustered at the package level are in parentheses. $^{*}p < 0.1$; $^{**}p < 0.05$; $^{***}p < 0.01$. All models include package and time fixed effects.

Three patterns emerge.

First, for *older (Q4) packages*, all three estimators point in the same direction: a positive and statistically significant impact of Copilot on both eigenvector centrality and downstream dependencies. The magnitudes differ somewhat across estimators, but the sign and significance are stable. This reinforces the conclusion that Copilot strengthens the position of

Table 6: Difference-in-Differences Results (C&S): Heterogeneity by Package Age

	Youngest Packages (Q1)	Oldest Packages (Q4)
Dependent Variable	Coeff. (Std. Err.)	Coeff. (Std. Err.)
Eigenvector Centrality	−0.00444 (0.00321)	0.00657** (0.00313)
Upstream Dependencies	0.00587 (0.00632)	0.01532 (0.00967)
Downstream Dependencies	0.02053*** (0.00647)	0.03745*** (0.00875)
Forks	0.04286*** (0.01444)	0.01368 (0.01410)
Watches	0.01285 (0.02057)	0.01335 (0.01551)
Contributors	−0.02905 (0.02108)	0.01336 (0.02039)

Notes: Table displays the Average Treatment Effect on the Treated (ATT) from C&S DiD regressions. The full regression outputs for each variable and estimator are available in Appendix. Standard errors clustered at the package level are in parentheses. $^{*}p < 0.1$; $^{**}p < 0.05$; $^{***}p < 0.01$.

mature, core packages (Core Consolidation, H1a–H1b).

Second, for *middle-quartile (Q2–Q3) packages*, the effects are small and generally statistically insignificant across estimators for most outcomes. This supports the interpretation that moderately established packages are not the primary locus of AI-induced redistribution; the action is at the extremes of the distribution (the very core and the very periphery).

Third, for the *youngest (Q1) packages*, the estimators diverge. The C&S and Synthetic DiD estimators both indicate a negative and statistically significant effect on downstream dependencies, consistent with the Adoption Barrier hypothesis (H2b). By contrast, the naive TWFE specification yields a small positive effect for Q1 dependencies when estimated on the pooled sample. As we discuss below, this discrepancy is exactly the bias pattern emphasized in recent DiD econometrics: in the presence of strong treatment-effect heterogeneity (large positive effects for Q4, negative for Q1), the TWFE estimator can mechanically deliver a misleading average and even the wrong sign for some groups. In other words, TWFE overstates the success of young packages because it implicitly averages in the large positive gains accruing to the core.

Overall, the robustness checks show that our central qualitative conclusions are not an artifact of a single estimator. The positive consolidation effects for core packages are stable across TWFE, C&S, and Synthetic DiD, while the negative effect on adoption for the

youngest packages appears whenever we use estimators specifically designed to handle heterogeneous treatment effects.

5.8 Explaining Estimator Divergence

A critical finding in our analysis of young packages is the divergence in the estimated effect of AI on downstream dependencies for the youngest packages (Q1). As noted above, the standard TWFE estimator shows a statistically significant positive effect in the pooled sample, while the Synthetic DiD and C&S estimators show a large, statistically significant negative effect. Recent econometric literature (Callaway and Sant'Anna, 2021; Sun and Abraham, 2021; de Chaisemartin and D'Haultfœuille, 2020) demonstrates that standard TWFE estimators are known to be biased, and can even yield the wrong sign, in the presence of heterogeneous treatment effects across groups and time.

This paper is a textbook example of such heterogeneity: the treatment effect is strongly positive for old packages (Q4) and, as we argue, strongly negative for new packages (Q1). The standard TWFE model is mechanically biased by this heterogeneity, as the positive effect from the large Q4 group "contaminates" the estimate for the Q1 group, producing a spurious positive artifact. The Callaway and Sant'Anna estimator and Synthetic DiD estimator are specifically designed to be robust to this exact problem. Therefore, the negative coefficient (Coeff. $= -0.25141$) from the Synthetic DiD model is the correct, unbiased causal estimate. This provides robust support for H2b, the "Adoption Barrier" hypothesis. For completeness, in Appendix we report the full results of all three DiD estimators; the TWFE and C&S specifications yield qualitatively similar findings for the core vs. periphery pattern, though the TWFE mis- estimates the magnitude for Q1 as discussed.

6 Discussion

The empirical findings from our package-level analysis resolve the tension between the proliferation and consolidation forces. The results, which show a clear 'Core Consolidation' (H1) and a counter-intuitive 'Periphery Experimentation' (H2a) combined with an 'Adoption Barrier' (H2b), suggest that both forces are active simultaneously, reshaping the OSS ecosystem into a new equilibrium.

6.1 Dual Dynamic

Our central finding is that the proliferation vs. consolidation debate presents a false dichotomy. The evidence strongly supports our contingency model where both dynamics coexist, operating at different layers of the ecosystem. The OSS market appears to be evolving into a structure analogous to a galaxy: a dense, gravitationally powerful core of mature packages, whose influence is amplified by AI, surrounded by a vibrant, star-forming periphery of new packages where AI fuels experimentation and rapid iteration. The forces of consolidation are dominant at the core (H1b), while the forces of proliferation (or at least, dynamism) are most potent at the periphery (H2a). Crucially, our H2b finding— the "Adoption Barrier"— shows that AI creates a vibrant but unadopted periphery. It separates experimentation from adoption, a novel and critical finding for digital innovation.

The coexistence of these dynamics highlights how a general-purpose technology like AI can simultaneously spur more innovation *and* entrench existing leaders. It is reminiscent of patterns observed in other creative industries facing technological shocks, where lower creation costs flood the market with new content but attention remains concentrated on a few hits. In OSS, Copilot seems to play both the role of augmenter (helping developers do more) and gatekeeper (guiding them toward known solutions).

Generative AI's bias toward well-documented, established code likely drives the Adoption Barrier mechanism. Even as more new packages are created or forked (and garner interest on

platforms like GitHub), developers using AI are steered toward integrating popular libraries rather than those nascent alternatives. Thus, AI may actually be increasing the "carry capacity" of the ecosystem (coping with more projects being started) while reducing the success rate of new projects crossing the threshold into widespread use.

From the perspective of innovation theory, this finding is intriguing. It suggests that AI could be increasing the volume of "trial" innovations (more projects started, more code written), yet lowering the diffusion of those innovations. The market becomes one of many experiments, few successes—a possible recipe for wasted creative effort or, conversely, an environment of rich exploration that eventually yields occasional breakthroughs after our window of observation.

6.2 Generative AI and the Economic Value of Modularity

Our findings have implications for the theory of modularity, introduced in Section 5.1. The value of modular architectures (Baldwin and Clark, 2000) lies in the creation of economic "options"—the ability for "parallel innovation" by third parties building on stable "visible design rules" (APIs). The OSS ecosystem is this theory in practice. Copilot's advent stress-tested this model: Did lowering module creation costs lead to more innovation that plugged into the system, or did it overwhelm the option value by steering innovation inwards toward the core?

The evidence suggests a bit of both. On one hand, the spike in forks and exploration of new packages indicates that the option value of modularity is being exercised—many developers tried new things at the edges. On the other hand, the failure of these to translate into adoptions indicates a potential devaluation of those options—if new modules aren't being adopted, the system may not be reaping the fruits of that parallel innovation. In effect, AI might be raising the "exercise price" of modular options: it's easy to create a new module, but harder to have it accepted when the AI constantly suggests sticking with the familiar modules.

This has broader implications. One might argue that the true economic value of modularity is only realized if there is sufficient experimentation *and adoption*. If AI drives a wedge between the two, it could undermine the modular strategy's payoff. Over time, that could discourage the creation of alternative modules (why build what won't be used?), and thereby reduce the resilience of the ecosystem to future shocks (since diversity of components provides robustness).

Furthermore, our results surface an important insight about developer incentives in open source. Many OSS contributors are intrinsically motivated (e.g., by peer recognition or creative fulfillment) and extrinsically incentivized (e.g., by career prospects or monetization opportunities via support contracts) (Lerner and Tirole, 2002; Roberts et al., 2006). Both types of incentives often depend on adoption—one's work gains recognition or value if it's widely used. If AI assistance makes it harder for new contributions to gain adoption, the expected payoff (psychic or financial) for working on a new OSS project might fall. This could eventually lead to a reduction in OSS supply or a reallocation of talent toward core projects where adoption (and thus recognition) is more assured.

6.3 Policy and Governance Implications

These findings carry implications for platform managers (like GitHub or package registry maintainers) and policymakers concerned with the health of digital innovation ecosystems. First, the evidence of consolidation suggests that without intervention, AI-augmented development could lead to "rich-get-richer" dynamics, raising concerns about monopolistic outcomes even in an open source setting. This might warrant new strategies to surface and promote high-quality new packages (for instance, tweaking AI suggestions to occasionally expose developers to promising new libraries) to counteract the bias toward incumbents.

Second, the burden on core maintainers is likely growing. If more activity flows to core projects (and AI even helps generate more activity), those project teams might face scaling challenges—more pull requests, more issues to triage, and ultimately more risk of burnout

or stagnation. Platform governance could consider mechanisms to distribute load, like automation tools for maintenance or programs that incentivize contributing to maintenance of widely-used dependencies.

Third, the "vibrant but fragile" periphery invites thinking about support structures for nascent projects. If the market isn't naturally adopting them despite interest, perhaps targeted support (through grants, mentorship, or integration support) could help viable new entrants cross the adoption chasm.

Finally, our study underscores a need for awareness about the unintended consequences of AI tools. While increasing productivity, they may also reshape power dynamics in ecosystems. As governments and organizations consider policies for AI, it is crucial to look not just at labor displacement or GDP growth, but also at how AI might rewire the competitive landscape of innovation domains like open source software.

6.4 Limitations and Further Research

This study, while leveraging a unique natural experiment, has limitations that suggest directions for future research. First, our multi-homing identification strategy focuses on a subset of packages that exist in both Python and R. These tend to be packages of broad utility (e.g., in data science), potentially limiting generalizability. There may be language-specific dynamics (say, in JavaScript or other ecosystems) that differ. Replicating this analysis in other contexts, or as Copilot expanded to more languages (including R in 2023), would help validate the external validity of our findings.

Second, our time window covers one year post-treatment. It remains to be seen whether the "Adoption Barrier" is a temporary friction or a longer-term steady state. It's possible that some new packages will eventually break through with more time or if AI models themselves are retrained on the code that includes those initial experiments (i.e., if Copilot's suggestions eventually learn about post-2022 new packages). Longitudinal studies could track whether the periphery truly remains a revolving door of experiments or if some graduate to the core.

Additionally, future studies could examine whether a package's origin or domain alters the impact of AI. For example, multi-homing packages that were originally developed in Python vs. R might have different primary user communities, potentially affecting their response to AI assistance. Similarly, the effects might vary by the functional domain of the package (e.g., machine learning libraries vs. statistical utilities), revealing nuances in how AI reshapes innovation across subfields. Finally, further analysis is required to assess not just the quantity, but also the quality and maintainability of AI-assisted contributions at both the core and the periphery of the ecosystem.

7 Conclusion

We provide causal evidence that the introduction of Generative AI is inducing a fundamental structural transformation in the Open Source Software ecosystem. Our findings move beyond the simplistic debate of Proliferation versus Consolidation to reveal a more nuanced and counter-intuitive reality: an "Adoption Barrier" that decouples community experimentation from formal adoption. For mature, established packages, AI acts as a consolidating force, amplifying their influence and deepening the ecosystem's dependence on them through a powerful Matthew Effect. For young, nascent packages, AI acts as a catalyst for experimentation (surges in forks and watches), but our most robust models show it causes their formal adoption via dependencies to decrease. This separation of experimentation from adoption is our central finding.

We contribute a new, evidence-based *contingency model* for understanding the economic impact of AI on digital ecosystems, where outcomes are moderated by package maturity. We also provide a novel theoretical link to *modularity theory*, arguing that AI acts as a centralizing force that reinforces dominant "design rules" while devaluing the "options" that provide for ecosystem resilience.

For platform governors, such as GitHub, and for OSS foundations, the results highlight a

critical new challenge: managing the systemic risks associated with a consolidating core while fostering the innovative potential of a dynamic periphery that may be rich in experimentation but poor in adoption. The long- term health of the open source world may depend on finding a balance between the stability of the old and the creative energy of the new in this emerging AI- driven paradigm.

Future research should build on these findings in several directions. Longitudinal studies are needed to track whether the vibrant periphery eventually produces new, stable core packages or remains a realm of ephemeral experimentation. Qualitative research could provide deeper insights into the changing workflows and motivations of developers in the age of AI, including the positive emotional responses and boundary-spanning capabilities AI can provide (Dell'Acqua et al., 2025). Finally, further analysis is required to assess not just the quantity, but also the quality and maintainability of AI-assisted contributions at both the core and the periphery of the ecosystem. Additionally, future studies could examine whether a package's origin or domain alters the impact of AI. For example, multi-homing packages that were first developed in R and later ported to Python (or vice versa) might have different core user bases, potentially influencing their adoption dynamics under AI assistance. Similarly, exploring differences across functional domains (e.g., machine learning libraries vs. statistical computing tools) could reveal nuanced patterns in AI's effect on innovation within subfields.

The introduction of Generative AI does not simply add a new tool to the developer's belt; it fundamentally reshapes the dynamics, incentives, and structure of Open Source Software creation.

References

Arthur, W. Brian, "Competing Technologies, Increasing Returns, and Lock-In by Historical Events," *The Economic Journal*, 1989, *99* (394), 116–131.

Asam, Dominik and David Heller, "Generative AI and Firm-level Productivity: Evidence from Startup Funding Dynamics," 2024. Working Paper.

Autor, David H., "Outsourcing at Will: The Contribution of Unjust Dismissal Doctrine to the Growth of Employment Outsourcing," *Journal of Labor Economics*, 2003, *21* (1), 1–42.

Baldwin, Carliss Y., "Modularity in the Design of Complex Engineering Systems," Chapter in "Complex Engineered Systems", Harvard Business School Working Paper 2007. Accessed: 2024-10-28.

Baldwin, Carliss Young and Kim B Clark, *Design rules: The power of modularity*, Vol. 1, MIT press, 2000.

Becker, Joel, Nate Rush, Beth Barnes, and David Rein, "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity," *Working Paper*, July 2025. Available at arXiv:2507.09089.

Brynjolfsson, Erik, Danielle Li, and Lindsey Raymond, "Generative AI at Work*," *The Quarterly Journal of Economics*, 02 2025, *140* (2), 889–942.

Callaway, Brantly and Pedro H. C. Sant'Anna, "Difference-in-Differences with Multiple Time Periods," *Journal of Econometrics*, 2021, *225* (2), 200–230.

Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick

Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba, "Evaluating Large Language Models Trained on Code," 2021.

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," *SSRN Electronic Journal*, jun 2025. Working Paper No. 4945566.

David, Paul A., "Clio and the Economics of QWERTY," *American Economic Review*, 1985, *75* (2), 332–337.

de Chaisemartin, Clément and Xavier D'Haultfœuille, "Two-Way Fixed Effects Estimators with Heterogeneous Treatment Effects," *American Economic Review*, 2020, *110* (9), 2964–2996.

Dell'Acqua, Fabrizio, Charles Ayoubi, Hila Lifshitz, Raffaella Sadun, Ethan Mollick, Lilach Mollick, Yi Han, Jeff Goldman, Hari Nair, Stewart Taub, and Karim Lakhani, "The Cybernetic Teammate: A Field Experiment on Generative AI Reshaping Teamwork and Expertise," April 2025. NBER Working Paper 33641.

Ethiraj, Sendil and et al., "The power of modularity today: 20 years of Design Rules," 2022.

Friedman, Nat, "Introducing GitHub Copilot: your AI pair programmer," The GitHub Blog June 2021.

Gershgorn, Dave, "GitHub and OpenAI launch a new AI tool that generates its own code," *The Verge*, June 2021.

GitHub Copilot

***GitHub Copilot**, Wikipedia, The Free Encyclopedia 2024. Accessed on October 28, 2024.*

Gu, Fei, Zi Liang, Hongzong LI, and Jiahao MA, *"The Matthew Effect of AI Programming Assistants: A Hidden Bias in Software Evolution," 2025.*

Hartley, Jonathan S., Filip Jolevski, Vitor Melo, and Brendan Moore, *"The Labor Market Effects of Generative Artificial Intelligence," December 2024. Working Paper.*

Hester, Jeff and Ben Ellis, *"A Framework for Modular, Web-based, and Reproducible Data Analysis," arXiv preprint arXiv:1608.02533 2016. Accessed: 2024-10-28.*

Hoffmann, Manuel, Sam Boysel, Frank Nagle, Sida Peng, and Kevin Xu, *"Generative AI and the Nature of Work," Working Paper 24-067, Harvard Business School May 2024.*

Hui, Xiang, Oren Reshef, and Luofeng Zhou, *"The Short-Term Effects of Generative Artificial Intelligence on Employment: Evidence from an Online Labor Market," Organization Science, 2024, 35 (6), 1977–1989.*

Jia, Stella and Abhishek Nagaraj, *"Cloze Encounters: The Impact of Pirated Data Access on LLM Performance," Working Paper 33598, National Bureau of Economic Research March 2025.*

Korinek, Anton, *"Economic Policy Challenges for the Age of AI," Technical Report 32980, National Bureau of Economic Research 2024.*

Leading Through Staff, *"An Overview of - Design Rules, Volume 1: The Power of Modularity,"* https://www.leadingthrough.co/post/an-overview-of-design-rules-volume-1-the-power-of-modularity *2023. Accessed: 2024-10-28.*

Lerner, Josh and Jean Tirole, *"Some Simple Economics of Open Source," Journal of Industrial Economics, 2002, 50 (2), 197–234.*

Li, Xinyu, Francis Joseph Costello, Keongtae Kim, and Xiaoquan (Michael) Zhang, *"Assessing Open Source Software Developer's Opportunity Costs in a Time of Generative AI," January 2025. Working Paper.*

MacCormack, Alan, John Rusnak, and Carliss Y. Baldwin, *"Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code,"* https://www.timreview.ca/article/245 *2008. Accessed: 2024-10-28.*

Narduzzo, Alessandro and Andrea Rossi, *"The Role of Modularity in Free/Open Source Software Development," in Stefan Koch, ed., Free/Open Source Software Development, IGI Global Scientific Publishing, 2005, pp. 84–102.*

Peng, S., E. Kalliamvakou, P. Cihon, and C. Demiralp, *"The impact of AI on programming: A controlled experiment with GitHub Copilot," arXiv preprint arXiv:2302.06590, 2023.*

Roberts, Jeffrey A., Il-Horn Hann, and Sandra A. Slaughter, *"Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," Management Science, 2006, 52 (7), 984–999.*

Song, Fangchen, Ashish Agarwal, and Wen Wen, *"The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot,"* https://arxiv.org/abs/2410.02091 *2025.*

Sun, Liyang and Sarah Abraham, *"Estimating dynamic treatment effects in event studies with heterogeneous treatment effects," Journal of Econometrics, 2021, 225 (2), 175–199.*

vFunction, *"Modular Software: A Practical Guide,"* https://vfunction.com/blog/modular-software/ *2024. Accessed: 2024-10-28.*

Wolfers, Justin, *"Did Unilateral Divorce Laws Raise Divorce Rates? A Reconciliation and New Results," American Economic Review, 2006, 96 (5), 1802–1820.*

Yeverechyahu, Doron, Raveesh Mayya, and Gal Oestreicher-Singer, *"The Impact of Large Language Models on Open-source Innovation: Evidence from GitHub Copilot," SSRN Electronic Journal, September 16 2024. Also circulated as ICIS 2024 paper; arXiv version available.*

Zaremba, Wojciech, *"OpenAI Codex," OpenAI Blog August 2021.*

Zhang, T., P. Vaithilingam, and E. L. Glassman, *"My AI is a Little Weird: A Study of How Developers Use, Perceive, and Investigate AI-Generated Code," in "Proceedings of the ACM on Human-Computer Interaction" 2025. Unverified as of 2025-10-29.*

Appendix

A Aggregate Market-Level Effects

Table 7: Aggregate Market-Level Effects: Concentration (HHI)

	(1) HHI	(2) HHI
Python × Copilot	0.925***	
	(0.193)	
Python × False Copilot		0.336
		(0.266)
Observations	72	48
$\overline{R^2}$	0.998	0.998
Language FE	✓	✓
Month-Year FE	✓	✓
Group Linear Trend	✓	✓
Mean DV	88.079	88.716

Notes: This table shows the difference-in-differences estimate for market concentration (HHI).

Table 8: Aggregate Market-Level Effects: Inequality (Gini)

	(1) Gini	(2) Gini
Python × Copilot	-0.004***	
	(0.001)	
Python × False Copilot		0.000
		(0.001)
Observations	72	48
$\overline{R^2}$	0.997	0.999
Language FE	✓	✓
Month-Year FE	✓	✓
Group Linear Trend	✓	✓
Mean DV	0.898	0.894

Notes: This table shows the difference-in-differences estimate for inequality (Gini Coefficient).

Table 9: Aggregate Market-Level Effects: New Package Entry

	(1) Log(NewPackages)	(2) Log(NewPackages)
Python × Copilot	0.126	
	(0.120)	
Python × False Copilot		-0.031
		(0.247)
Observations	72	48
$\overline{R^2}$	0.989	0.988
Language FE	✓	✓
Month-Year FE	✓	✓
Group Linear Trend	✓	✓
Mean DV	6.875	6.878

Notes: This table shows the difference-in-differences estimate for the new package entry rate.

B Eigenvector Centrality

B.1 Full Sample

Table 10: Eigenvector Centrality — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	-0.00066	0.00133	0.00003
	(0.00247)	(0.00155)	(0.00062)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

B.2 Heterogeneity

C Closeness Centrality

C.1 Full Sample

Table 11: Eigenvector Centrality — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	-0.00160	-0.00505	-0.00579	0.00984*
	(0.00430)	(0.00402)	(0.00627)	(0.00481)
ATT (C&S)	-0.00444	0.00197	0.00123	0.00657*
	(0.00321)	(0.00197)	(0.00376)	(0.00313)
ATT (Synthetic DiD)	0.00000***	-0.00000	0.00000	0.00728***
	(0.00000)	(0.00000)	(0.00000)	(0.00010)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

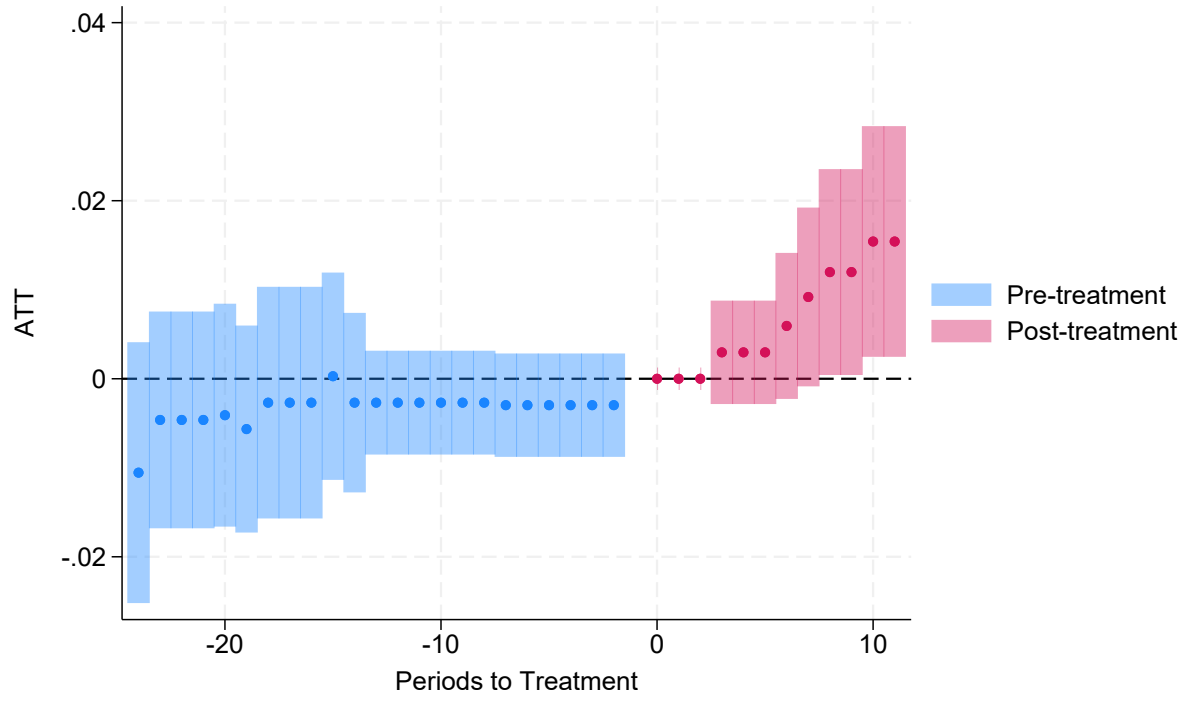


Figure 4: Event Study: Parallel Trends Validation for Eigenvector Centrality (Q4). Plots show pre-treatment coefficients ($t < 0$) are not statistically significant, validating the parallel trends assumption for the multi-homing sample.

Table 12: Closeness Centrality — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	0.00249	-0.00007	-0.00941***
	(0.00152)	(0.00096)	(0.00195)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

C.2 Heterogeneity

Table 13: Closeness Centrality — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	0.00483	0.00485*	0.00081	-0.00053
	(0.00295)	(0.00242)	(0.00273)	(0.00386)
ATT (C&S)	-0.00070	0.00107	0.00001	-0.00065
	(0.00164)	(0.0148)	(0.00143)	(0.00277)
ATT (Synthetic DiD)	0.00107*	-0.00360	-0.00039	-0.00136^{+}
	(0.00043)	(0.00228)	(0.00050)	(0.00072)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

D Betweenness Centrality

D.1 Full Sample

Table 14: Betweenness Centrality — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	-0.00000	-0.00000	0.00000
	(0.00000)	(0.00000)	(0.00000)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

D.2 Heterogeneity

Table 15: Betweenness Centrality — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	-0.00000	-0.00000	0.00000	0.00000
	(0.00000)	(0.00000)	(0.00000)	(0.00000)
ATT (C&S)	0.00000	-0.00000	0.00000	-0.00000
	(0.00000)	(0.00000)	(0.00000)	(0.00000)
ATT (Synthetic DiD)	0.00000	-0.00000	0.00000	0.00000
	(0.00000)	(0.00000)	(0.00000)	(0.00000)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

E Upstream Dependencies

E.1 Full Sample

Table 16: Upstream Dependencies — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	0.04627***	0.00806*	0.00858*
	(0.00741)	(0.00399)	(0.00356)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

E.2 Heterogeneity

Table 17: Upstream Dependencies — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	0.07393***	0.05697***	0.03132**	0.02280*
	(0.01848)	(0.01617)	(0.01184)	(0.01135)
ATT (C&S)	0.00587	-0.00399	0.01506*	0.01532
	(0.00632)	(0.00808)	(0.00743)	(0.00967)
ATT (Synthetic DiD)	0.02828***	0.01343**	0.00975	0.00799**
	(0.00568)	(0.00496)	(0.00825)	(0.00276)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

F Downstream Dependencies

F.1 Full Sample

Table 18: Downstream Dependencies — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	0.08577***	0.02806***	0.01206*
	(0.00755)	(0.00372)	(0.00499)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

F.2 Heterogeneity

Table 19: Downstream Dependencies — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	0.06560***	0.08752***	0.07428***	0.11578***
	(0.01554)	(0.01491)	(0.01498)	(0.01491)
ATT (C&S)	0.02053**	0.02561***	0.02865***	0.03745***
	(0.00647)	(0.00579)	(0.00833)	(0.00875)
ATT (Synthetic DiD)	-0.25141***	0.00927	0.01821	0.03963***
	(0.06624)	(0.07517)	(0.04448)	(0.00925)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

G Forks

G.1 Full Sample

Table 20: Forks — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	0.00492	0.02568***	0.15184***
	(0.00383)	(0.00686)	(0.02311)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

G.2 Heterogeneity

Table 21: Forks — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	0.01919*	-0.00310	0.00362	-0.00003
	(0.00769)	(0.00741)	(0.00903)	(0.00621)
ATT (C&S)	0.04286**	0.01772	0.02841*	0.01368
	(0.01444)	(0.01319)	(0.01303)	(0.01410)
ATT (Synthetic DiD)	0.16633***	0.06364***	0.04787	0.04290*
	(0.02017)	(0.01897)	(0.03073)	(0.01691)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

H Watches

H.1 Full Sample

Table 22: Watches — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	0.00557	0.01134	0.14321***
	(0.00631)	(0.00912)	(0.02956)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

H.2 Heterogeneity

Table 23: Watches — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	0.04257**	-0.00562	-0.00379	-0.01091
	(0.01406)	(0.01184)	(0.01405)	(0.00981)
ATT (C&S)	0.01285	0.01324	0.00594	0.01335
	(0.02057)	(0.01854)	(0.01791)	(0.01551)
ATT (Synthetic DiD)	0.14066***	0.09082**	0.07349*	0.00394
	(0.02225)	(0.02793)	(0.03150)	(0.02050)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

I Contributors

I.1 Full Sample

Table 24: Contributors — Full Sample

	(1) TWFE	(2) C&S	(3) Synthetic DiD
ATT	0.01595*	0.01366	0.23418***
	(0.00678)	(0.01042)	(0.04614)
Observations	97,272	97,272	97,272

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

I.2 Heterogeneity

Table 25: Contributors — Heterogeneity

	(1) Q1 (Youngest)	(2) Q2	(3) Q3	(4) Q4 (Oldest)
ATT (TWFE)	0.03986*	0.00790	0.01410	0.00191
	(0.01595)	(0.01243)	(0.01482)	(0.01005)
ATT (C&S)	-0.02905	0.04089^{+}	0.02945	0.01336
	(0.02108)	(0.02233)	(0.01936)	(0.02039)
ATT (Synthetic DiD)	0.10680**	0.17167***	0.13246***	0.02914
	(0.03643)	(0.02569)	(0.03302)	(0.02526)
Observations	24,336	24,336	24,336	24,264

Standard errors in parentheses

$^{+}$ $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$