Pre/Post LLMs For Code Era: A study based on Github Copilot

1st Aya Garryyeva
Department of Computer Science)
William & Mary
Williamsburg, VA
lgarryyeva@wm.edu

2nd Mostafa Ahmed
Department of Computer Science)
William & Mary
Williamsburg, VA
email address or ORCID

***Abstract*—GitHub Copilot, an AI-pair programming assistant introduced in June 2021, has rapidly become a prominent tool in software development, with widespread adoption across individual developers, enterprise customers, and open-source projects. This study investigates the impact of Copilot on open-source software development practices by analyzing trends in commit frequency, author participation, and code churn across repositories with confirmed usage of Copilot. Using an interrupted time series framework, we assess longitudinal patterns to evaluate whether Copilot's adoption aligns with changes in software development practices. Our findings reveal that while commit frequency remained largely unchanged, author participation demonstrated a gradual increase post-Copilot, suggesting growing community engagement. Additionally, an initial rise in commit churn after Copilot's launch points to potential short-term adjustments in developer behavior. These results highlight the nuanced influence of AI tools like Copilot in shaping collaborative development environments.**

***Index Terms*—Large Language Models, Github Copilot, Software Engineering**

I. Introduction

GitHub Copilot was introduced in June 2021 as a new AI pair programmer designed to aid developers in writing better code [1]. Initially offered as a technical preview to a limited number of users, Copilot garnered significant attention, with over 1.2 million developers participating in the preview during the following 12 months [2]. By June 2022, GitHub announced Copilot's general availability for individual developers, extending access to enterprise customers by December 2022. Within months, over 400 organizations had adopted the tool, a figure that rose to over 20,000 organizations by mid-2023 [3]. This rapid expansion was accompanied by an increase in supported programming languages and a commitment to provide free access for students and maintainers of popular open-source repositories [4].

The introduction of Copilot represents a significant milestone in the integration of AI tools into software development workflows. With its ability to suggest contextually relevant code, Copilot promises to enhance productivity, reduce cognitive load, and potentially reshape collaborative programming practices [5]. However, despite its rapid adoption, the effects of Copilot on core development metrics, such as code committing behavior, contributor participation, and code modification patterns, remain under-explored. Understanding these impacts is essential to assess the broader implications of AI tools in open-source ecosystems, particularly in fostering inclusivity and efficiency.

In this study, we focus on popular open-source repositories, anticipating their eligibility for free Copilot access, to investigate how AI-assisted coding influences software development practices. Using metrics such as commit frequency, author trends, and commit churn, we aim to uncover whether Copilot's adoption aligns with meaningful shifts in development behavior. Through this analysis, we seek to provide a foundational understanding of Copilot's role in shaping the evolving landscape of open-source development.

II. Related Works

Several studies have investigated questions related to the impact of AI tools, such as GitHub Copilot, on software development practices. Song et al. [6] examined how GitHub Copilot affects productivity and participation in open-source projects. Here, productivity is measured in merged pull requests and individual participation in number of pull request authors. They found Copilot improved project-level productivity by 6.5% and individual participation by 5.4%. However, they also noted a 41.6% increase in integration time, highlighting the need for coordination among developers. The study suggests that Copilot's benefits vary based on developers' familiarity with projects, particularly aiding core contributors. This research provides a detailed understanding of Copilot's role in collaborative development settings. In contrast to our study, the researchers capture the data between January 2021 and December 2022. Additionally, the study was conducted in collaboration with GitHub and utilized proprietary data that the company has provided.

Vaithilingam et al. [7] evaluate the usability of GitHub Copilot. The researchers conducted a within-subjects user study with 24 participants performing programming tasks in Python to evaluate Copilot's impact on programming workflows. The study found that while many participants preferred Copilot for daily programming because it offered helpful starting points, Copilot did not consistently improve task completion times or success rates.

Mozannar et al. [8] explore how developers interact with GitHub Copilot and other code-recommendation systems. The

study focuses on understanding the productivity gains and challenges associated with such tools, using a taxonomy called CodeRec User Programming States (CUPS). CUPS is a framework for categorizing developer activities when interacting with AI tools, which includes states like "Thinking/Verifying Suggestions," "Prompt Crafting," "Debugging/Testing Code," and "Editing Suggestions." As part of the study 21 programmers performed coding tasks using Copilot and retrospectively labeling their actions. The findings indicate that Copilot accelerates routine tasks but introduces new efforts, such as refining prompts and verifying suggestions. Specifically, developers spend significant time in states like "Thinking/Verifying" (22.4%) and "Editing Suggestions" (11.9%).

Some of the studies were conducted by GitHub's internal research teams. Ziegler et al. [5] examine how tools like GitHub Copilot impact software developers' productivity. The study investigates the extent to which developers perceive productivity improvements through feedback mechanisms, especially focusing on the rate of accepted suggestions as a primary productivity indicator. Similar to our study the most common language among the user base were TypeScript, JavaScript, and Python with the latter two having higher acceptance rates. Overall, the study concluded that the acceprance rate of the code suggestions is highly correlated with the self-reported productivity of the developers. The study does not conduct an empirical evaluation of the overall effect of Copilot on open-source software development practices.

Peng et al. [9] present findings from a controlled experiment exploring how GitHub Copilot affects software developers' productivity. The study was a result of collaboration by researchers at Microsoft Research and GitHub. In a randomized control trial (RCT) setting 95 participants were asked to implement an HTTP server in JavaScript. The treated group utilized Copilot, while the control group relied on conventional tools such as Stack Overflow and internet searches. Performance metrics included task success rates and completion times, tracked via GitHub Classroom. The findings indicate that developers with access to GitHub Copilot completed programming tasks 55.8% faster than those without the tool. Moreover, the productivity gain was most pronounced among developers with less experience, older programmers, and those coding more hours daily.

These studies complement our research focus by providing empirical evidence of Copilot's influence on productivity, developer behavior, and project outcomes in real-world settings.

III. Methodology

We collected and statistically analyzed data from a sample of open-source projects on Github that adopted Github Copilot.

A. Data Collection

Our data collection effort was a three-stage process. First, we collected a comprehensive list of repositories that use eight of the programming languages supported by Copilot. From that list, we extracted the most popular repositories and performed a keyword search from commits, pull requests, and issues. Next, we manually analyzed the keyword search results to identify the repositories where Copilot is mentioned and explicitly indicate usage of Copilot. Finally, using the list of repositories that are confirmed to use Copilot, we conducted aggregate data collection for different metrics in 30-day windows, 12 on each side around the launch of Github Copilot.

*1) **Initial Candidate Project Extraction**:* As a first step, we conducted a comprehensive data collection process to analyze repositories across multiple programming languages using the SEART GitHub Search (GHS) tool [10]. Our focus was on identifying repositories that are popular within the developer community, as popularity is a key factor for GitHub when determining which projects receive free access to Copilot for their maintainers.

Figure 1 illustrates the initial data collection step. Below we outline our data collection methodology, the filters applied, and the characteristics of the final dataset used for keyword search.

Fig. 1. Initial candidate project extraction for keyword search

We began by utilizing the GHS to identify repositories with primary programming languages that Github Copilot has supported from early days: Python, JavaScript, TypeScript, Java, C++, C#, Go, and Ruby. The initial selection criteria included:

- **Non-fork repositories** to ensure originality of content.
- **Minimum 100 commits** to ensure that the projects were actively maintained at some point.
- **Minimum 300 stars** allowing us to focus on repositories with significant popularity and usage.

The initial query yielded approximately 61,229 repositories across the selected seven programming languages.

Data Cleaning and Filtering

To refine the dataset, we applied a series of filters to focus on repositories that were both active and representative of popular projects. The cleaning and filtering steps included:

1) Select repositories where the last commit was after July 2022 to ensure that the projects were still active.

2) Focus on repositories with more than 10 contributors, suggesting community involvement and project robustness.
3) Exclude the repositories with YouTube channels as the homepage - these were often tutorials, not intended to be full-fledged projects.
4) Exclude projects owned by companies such as Microsoft, Google, GitHub, Amazon, and Facebook. This exclusion allows us to avoid data bias from large enterprise-driven projects of companies that develop LLM-based coding assistants themselves.
5) Select the repositories that were in top 50th Percentile for number of stars to pick well-regarded projects.

This rigorous filtering process resulted in a final set of $10,290$ repositories that met all the criteria.

*2) **Keyword Search and Manual Analysis**:* We conducted targeted searches through GitHub's API for 'Copilot' mentions in commits, capturing metadata including commit messages, author names, URLs, and dates. We extended this search to issues and pull requests, collecting titles, URLs, states (open/closed), and contextual mentions. To manage GitHub API's rate limitations effectively, our data collection scripts implemented robust error handling and rate monitoring, using dynamic sleep intervals and comprehensive error logging.

Our data collection process employed pagination and chronological ordering through GitHub API, retrieving items from oldest to newest, with each repository limited to 5 pages of 100 items each. This approach ensured comprehensive temporal coverage while managing API constraints, targeting the first evidence of Copilot usage in each repository.

While the initial search yielded substantial results, we identified records containing Chinese characters that required filtering to ensure reliable manual analysis. After applying Unicode-based filtering, we retained 559 commits, 1,300 issues, and 17,510 pull requests.

Our initial manual analysis revealed that not all Copilot mentions indicated actual tool usage. After thoroughly examining commit messages and pull request contexts, and removing duplicate repositories, we identified 241 repositories with verified Copilot adoption for our empirical study.

*3) **Aggregate Data Collection**:* Once we determined the 241 projects that ever used Copilot we utilized PyDriller Python package to capture aggregated monthly data metrics. We aim to capture detailed historical data over 30-day windows between July 1, 2020 and July 31, 2022 (to ensure 12 months on each side around the Github Copilot launch in late June 2021). To account for the transition period we excluded one month of data centered around launch of Copilot (July 2021).

Measures: For each repository, we collected global and time-window measures listed below:

- **main programming language**
- **project age** at the time of Copilot launch measured in months from the day the repository was created, capturing maturity effects..
- **project size** measured in total number of commits in projects' history, reflecting development activity levels.
- **number of non-merge commits** per time window to account for the recorded local changes made to the repository by the developer. These are identified as the commits with a single parent. Non-merge commits are a proxy for developer productivity and represent individual contributions to the codebase.
- **number of merge commits** per time window to capture the instances when two or more development histories were merged together. These are identified as the commits with more than one parent. Merge commits are a proxy for developer contribution and reflect collaborative efforts involving code integration.
- **number of commit authors** per time window as proxy of the project's community and accessibility of the open-source environment to the new comers. These authors include those with and without the write access.
- **mean commit churn** per time window to measure how much code was under change during that time. We calculate churn by adding the number of lines inserted and the number of lines deleted per commit. Churn could provide insights into code stability and work patterns.

Filtering Due to the presence of the outliers, we filter out the projects in top 1 percentile. As a result we end up with monthly aggregate data for 193 repositories.

B. Research Questions

In this study we pose three research questions to investigate the impact of the AI pair programming assistant, Copilot, on the software development practices:

- **RQ1:** Has launch of Copilot encouraged more frequent code committing?
- **RQ2:** Do open-source projects show increased newcomer participation?
- **RQ3:** What changes occurred in commit sizes since Copilot's launch?

The first question aims to understand whether the developers are committing code more frequently when the new AI pair programming tool is utilized. Commit frequency often used to measure productivity and could signify ease in addressing coding tasks. Copilot's utilization could streamline programming tasks, encouraging smaller and more frequent commits. However, if the developers trust the AI tools and rely more on the auto-generated code and summaries, they might commit less frequently.

The second question aims to gauge whether use of Copilot lowers the entry barrier for coding and expand the project communities on Github. Since Copilot provides contextual suggestions and completes code snippets, it may attract more newcomers to open-source projects

The third question aims to understand whether the developers' work structure has changed since introduction of Copilot.

Smaller commits may indicate a focus on incremental changes, while larger commits might suggest bulk implementation or reliance on auto-generated code. Hence, changes in the size of commits can proxy shifts in developer workflows, possibly influenced by Copilot's code generation capabilities.

C. Mixed Effect Model

The primary goal of this study is to evaluate the longitudinal effects of GitHub Copilot's launch on open-source software development activities. Specifically, we aim to analyze patterns in merge commits, non-merge commits, unique authors, and mean commit churn aggregated monthly across open-source repositories. We utilize a mixed-effects interrupted time series (ITS) model to capture both temporal trends and variability across repositories and programming languages. The study spans 24 months: 12 months prior to and 12 months after the launch of Copilot, with the month of Copilot's launch excluded as an adjustment period.

To answer the research questions our response variables include number of merge commits, non-merge commits, unique authors, and mean commit churn per time window. We fit an ITS model for each response variable to analyze both the level and slope changes following Copilot's launch. The general form of the model is:

$$Y_i = \beta_0 + \beta_1 \cdot \text{Time}_i + \beta_2 \cdot \text{Copilot}_i + \beta_3 \cdot \text{Time\_since\_Copilot}_i + \epsilon_i,$$

where *Time* represents the time in months relative to the observation period, capturing temporal trends; *Copilot* is a binary indicator (0 = pre-launch, 1 = post-launch) to measure the intervention's effect; and *Time since Copilot* counts months after Copilot's launch (0 before intervention) to capture post-launch trend changes.

Because our data is nested with each language and repository contributing contributing multiple observations, we implement the ITS model as a mixed-effect linear regression using *mixedlm* function within Python's *statsmodel* package. We include random effects terms for programming language and repository, while controlling for project size and age which are the confounding variables. We model all other variables as fixed effects. This will allow us to accounts for unobserved heterogeneity between repositories and capture variability attributable to programming languages.

Prior to fitting the model we standardize all numerical features since some of our features have different scales (e.g. time vs project size). Standardizing the features ensures that the influences of various features in the model are balanced. We assess the model's fit using **marginal R-squared** (R^2_m) to measure variance explained by fixed effects (e.g., Copilot and time trends) and **conditional R-squared** (R^2_c) to account for both fixed and random effects, indicating the model's overall explanatory power.

IV. Results and Discussion

In this section we complete exploratory study of the number of repositories representing each programming languages and the trends in the response variable plotted in the 24 month period. The remainder of the section describe the results for each model. We present the model coefficients and corresponding standard errors, and the P-values along with the the statistical significance is indicated by stars. We use these results to answer the research questions posed in this study.

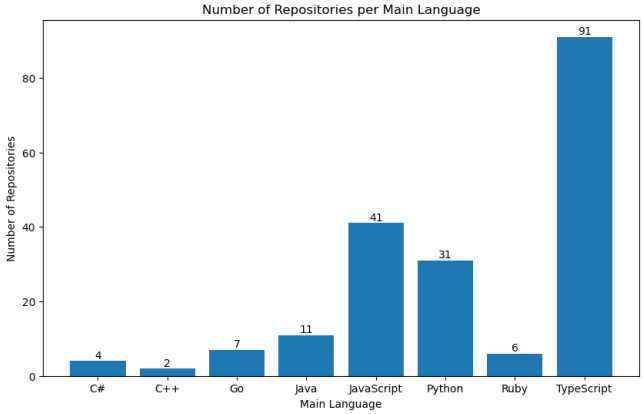

Fig. 2. Projects per Language

A. Exploratory Data Analysis

Figure 2 demonstrates the breakdown of the projects per main programming language. TypeScript, JavaScript, and Python are the top three languages represented in our data followed by Java. Unfortunately, Go, Ruby, C#, and C++ all have below 7 projects which could be insufficient for ensuring that there is enough variance during the modeling step of our analysis.

Based on the Figure 3 we can see a clear upward trend for the number of authors since launch of the Copilot (red dashed line). There are not obvious trends for other variables. We can also observe the 95% confidence interval for each feature around the lineplot, which indicate high levels of variability in the underlying data. Hence, the statistical analysis of the data will lead to a better understanding of the impact Copilot might have had on the metrics we have collected.

B. RQ_1: Commit Frequency Trends

Table I shows the output of the non-merge commit frequency model. The coefficients of determination R^2_m = 0.72, R^2_c = 0.67, indicate that the fixed effects explain a substantial portion of the variance, with additional variance captured by random effects. The **Copilot** intervention and **Time Since Copilot** were not statistically significant, suggesting no immediate or gradual change in non-merge commit frequency after Copilot's launch. **Project Size** had a strong positive effect, reflecting that larger projects contribute more to non-merge commit activity. **Project Age** showed a significant negative effect, indicating that older projects tend to have fewer non-merge commits per month.

Based on the II the merge commit model achieved an R^2_m = 0.76, R^2_c = 0.64, similarly showing a strong fit.

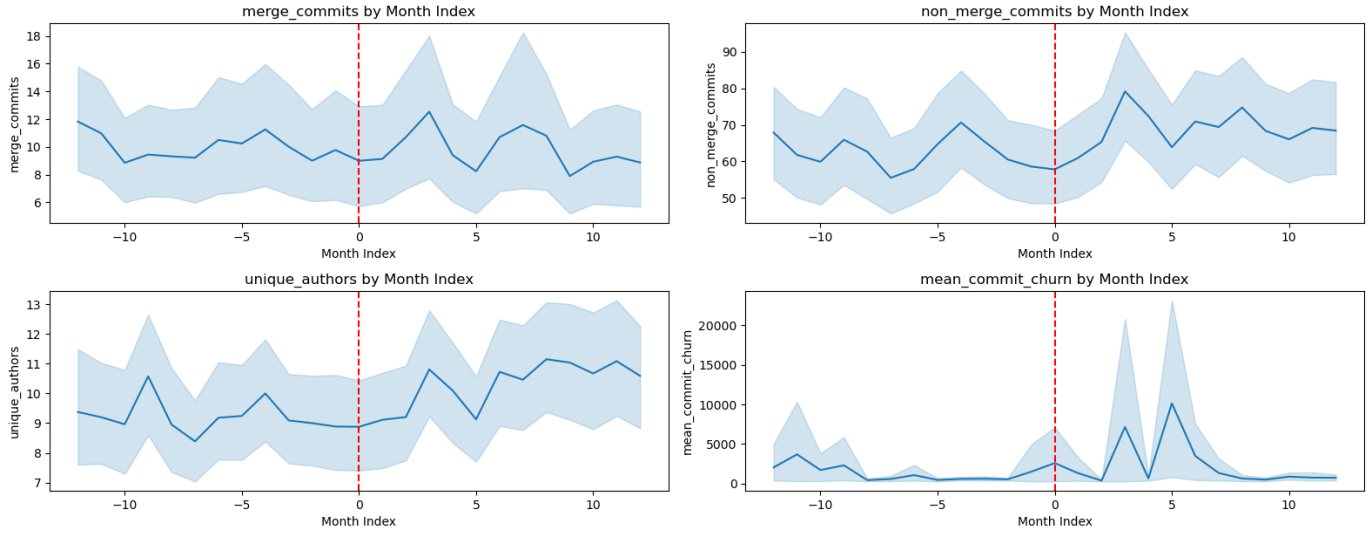

Fig. 3. Trends before and after the Github Copilot launch by Number of Merge Commits, Number of Non-Merge Commits, Number of Authors, and Mean Commit Churn.

The **Copilot** intervention and **Time Since Copilot** were not significant predictors of merge commit activity, suggesting no detectable shift in merge behavior post-Copilot. **Project Size** was again a significant positive predictor, consistent with larger projects driving more merge activity. **Project Age** approached significance, with older projects showing a potential decline in merge commit frequency.

Discussion: The results from the tables I and II provide several insights into commit frequency trends surrounding the launch of GitHub Copilot. Firstly, neither merge nor non-merge commit frequencies showed statistically significant changes associated with Copilot's introduction, suggesting that Copilot's influence may not directly manifest as changes in commit frequency. This could indicate that its effects might be more nuanced, such as improving code quality, developer experience, or workflow efficiency rather than increasing commit activity.

Project Age negatively correlated with non-merge commits and marginally with merge commits, reflecting that older projects may become more stable or experience reduced activity over time. **Project Size** consistently emerged as a significant driver of commit frequency, underscoring the importance of repository scale in understanding development activity.

It is important to note that the analysis relies on commit frequency as a proxy for developer activity, which may not fully capture shifts in how developers interact with AI-assisted tools, in our case Copilot. Moreover, the absence of significant effects might reflect a short observation window or variation in Copilot adoption rates across repositories.

RQ1: The models show no statistically significant increase in either merge or non-merge commit frequencies following Copilot's launch, suggesting that it did not directly encourage more frequent code committing.

C. RQ_2: Author Trends

Table III shows the results of the model analyzing the number of authors per month. They demonstrate a strong fit, with an $R^2_m = 0.86$, $R^2_c = 0.75$, indicating that fixed effects explain most of the variance, with additional variability captured by random effects. **Copilot** intervention was not a significant predictor, suggesting no immediate increase in the number of authors following Copilot's launch. However, **Time Since Copilot** was highly significant, indicating a positive trend in author participation post-Copilot, potentially reflecting

TABLE I
NON-MERGE COMMITS FREQUENCY MODEL. THE RESPONSE IS NUMBER OF NON-MERGE COMMITS PER MONTH. $R^2_m = 0.72$, $R^2_c = 0.67$.
Note: $^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$.

	Coefficients	Std. Err.	P-value
(Intercept)	-0.023	0.037	0.538
CopilotTRUE	0.052	0.033	0.116
Time	0.004	0.023	0.876
Time Since Copilot	0.008	0.018	0.648
*Age****	-0.231	0.035	0.000
*Size****	0.767	0.035	0.000

TABLE II
MERGE COMMITS FREQUENCY MODEL. THE RESPONSE IS NUMBER OF MERGE COMMITS PER MONTH. $R^2_m = 0.76$, $R^2_c = 0.64$.
Note: $^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$.

	Coefficients	Std. Err.	P-value
(Intercept)	-0.019	0.055	0.734
CopilotTRUE	0.040	0.030	0.185
Time	-0.029	0.021	0.171
Time Since Copilot	-0.008	0.017	0.616
Age	-0.103	0.056	0.064
*Size****	0.514	0.056	0.000

growing engagement over time. **Project Size** was a strong positive predictor, highlighting the role of larger repositories in attracting more contributors. **Project Age** was significant and negatively associated with author count, suggesting older projects might see less frequent participation from new or additional contributors.

TABLE III
AUTHOR MODEL. THE RESPONSE IS NUMBER OF AUTHORS PER MONTH.
R_m^2 = 0.86, R_c^2 = 0.75.
Note: ***p < 0.001, **p < 0.01, *p < 0.05.

	Coefficients	Std. Err.	P-value
(Intercept)	-0.011	0.049	0.820
CopilotTRUE	0.024	0.024	0.318
Time	-0.009	0.017	0.580
*Time Since Copilot****	0.053	0.013	0.000
*Age***	-0.136	0.050	0.006
*Size****	0.692	0.050	0.000

Discussion: The lack of significance for the immediate Copilot intervention effect suggests that Copilot did not result in an abrupt increase in author participation. However, the significant and positive Time Since Copilot coefficient suggests a gradual rise in contributor activity over time, potentially as adoption of Copilot spread or its effects became more apparent. The post-Copilot trend indicates that while Copilot may not have directly spurred immediate participation, its presence aligns with an upward trajectory in author engagement over time, which could reflect its influence on reducing barriers to contribution.

Project Size was a dominant factor in driving author counts, consistent with larger projects offering more opportunities for contributions. The negative effect of **Project Age** aligns with the idea that older projects may be more stable or less dynamic, attracting fewer new authors.

These findings suggest that while Copilot may not immediately alter the number of contributors, it aligns with trends of increased participation over time. This might point to indirect benefits of AI tools in making coding more accessible or lowering the effort required for onboarding and contributing.

RQ2: The author trends model indicates a significant positive increase in author participation over time after Copilot's launch, aligning with gradual growth in newcomer or contributor activity.

D. RQ_3: Code Churn Trends

The model analyzing mean commit churn per month exhibits modest explanatory power, with an R_m^2 = 0.08, R_c^2 = 0.07, suggesting limited variance explained by the fixed and random effects. **Copilot** intervention was the only statistically significant variable, indicating an immediate increase in mean commit churn following Copilot's launch. This could reflect an increase in developers experimenting with Copilot-generated code or larger edits driven by its suggestions. However, the lack of significance for Time Since Copilot indicates that this effect was not sustained or amplified over time.

Discussion: The absence of significant time effects indicates no detectable long-term trends in commit churn, either pre- or post-Copilot. Hence, any impacts on churn are more likely attributable to specific events or behaviors rather than sustained temporal changes. Unlike previous models, neither **Project Size** nor **Project Age** had significant effects on commit churn. This may suggest that churn, as a measure of code edits, is less influenced by repository size or maturity compared to metrics like commit frequency or author count.

The increase in churn following Copilot's launch might reflect shifts in developer behavior, such as trying more extensive code modifications or relying on Copilot for larger-scale changes. However, the short-lived nature of this effect raises questions about whether developers adjusted their workflows or if Copilot's impact normalized over time.

Limitations: The low coefficients of determination values suggest that commit churn is influenced by other factors not included in the model. (e.g. type of project, individual developer habits, or external events). As churn reflects code edits, further investigation could examine whether these edits were productive (e.g., bug fixes, feature additions) or experimental.

TABLE IV
COMMIT CHURN MODEL. THE RESPONSE IS MEAN COMMIT CHURNS PER MONTH. R_m^2 = 0.08, R_c^2 = 0.07.
Note: ***p < 0.001, **p < 0.01, *p < 0.05.

	Coefficients	Std. Err.	P-value
(Intercept)	-0.070	0.038	0.067
CopilotTRUE	0.137	0.060	0.022
Time	-0.035	0.041	0.391
Time Since Copilot	-0.022	0.033	0.504
Age	0.039	0.025	0.110
Size	-0.017	0.025	0.500

RQ3: The commit churn model reveals a statistically significant immediate increase in churn following Copilot's launch, but no sustained temporal changes, highlighting potential short-term shifts in developer behavior or code modification patterns.

V. THREATS TO VALIDITY

Due to laborious nature of the manual analysis, our dataset contains relatively low number of projects to guarantee robustness of the models.

Firstly, we are facing a threat to validity because of the inflation of zero values in our data leaving us unable to avoid biases in our conclusions. Th zero value inflation is because of the inconsistent activity of about 7% of projects during the 24-month observation period. Once we have enough projects in our data, these projects can be omitted from the analysis to remedy the issue.

Additionally, multivariate regression analysis, that we employ in this study, requires enough variance across all the modeled dimensions. Unfortunately, our data does not capture enough projects for some of the programming languages as mentioned above. Hence, our study needs to be expanded to add additional projects to fix this potential threat to validity.

Lastly, our commit churn model has low coefficients of determination values which means that the features we include in the model are not explaining the variability in the data. Hence, further analysis needs to be conducted in order to improve the model performance and gain meaningful outcomes of this model.

VI. Conclusion

This study explored the effects of GitHub Copilot's launch on open-source software development by analyzing trends in commit frequency, author participation, and code churn. The findings suggest that Copilot's introduction did not significantly influence the frequency of commits, whether merge or non-merge, indicating that its adoption may not directly alter developers' coding or committing patterns. However, the results showed a notable positive trend in author participation over time, suggesting that Copilot might lower barriers to entry for contributors, enabling a gradual increase in open-source engagement. Additionally, the significant short-term rise in commit churn following Copilot's launch hints at potential changes in how developers interact with the tool, possibly experimenting with larger or more complex edits in their codebases.

Overall, while Copilot's immediate effects on measurable coding activity were limited, the observed gradual increase in author engagement and short-term churn changes provide insights into its potential to shape software development practices. These findings highlight the importance of examining nuanced and indirect impacts of AI tools on collaborative development. Future research could investigate long-term effects, evaluate Copilot's influence on code quality or productivity, and explore adoption trends across diverse developer communities to better understand its role in transforming open-source ecosystems.

References

[1] N. Friedman. Introducing GitHub copilot: your AI pair programmer. [Online]. Available: https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[2] T. Dohmke. GitHub copilot is generally available to all developers. [Online]. Available: https://github.blog/news-insights/product-news/github-copilot-is-generally-available-to-all-developers/

[3] Copilot · GitHub changelog. [Online]. Available: https://github.blog/changelog/label/copilot/

[4] Getting free access to copilot as a student, teacher, or maintainer. [Online]. Available: https://docs.github.com/en/copilot/managing-copilot/managing-copilot-as-an-individual-subscriber/managing-your-copilot-subscription/getting-free-access-to-copilot-as-a-student-teacher-or-maintainer

[5] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Productivity assessment of neural code completion," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. ACM, pp. 21–29. [Online]. Available: https://dl.acm.org/doi/10.1145/3520312.3534864

[6] F. Song, A. Agarwal, and W. Wen, "The impact of generative AI on collaborative open-source software development: Evidence from GitHub copilot." [Online]. Available: https://papers.ssrn.com/abstract=4856935

[7] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. ACM, pp. 1–7. [Online]. Available: https://dl.acm.org/doi/10.1145/3491101.3519665

[8] H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz, "Reading between the lines: Modeling user behavior and costs in AI-assisted programming." [Online]. Available: http://arxiv.org/abs/2210.14306

[9] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of AI on developer productivity: Evidence from GitHub copilot." [Online]. Available: http://arxiv.org/abs/2302.06590

[10] GitHub search. [Online]. Available: https://seart-ghs.si.usi.ch/