Generative AI for Pull Request Descriptions: Adoption, Impact, and Developer Interventions

TAO XIAO, Nara Institute of Science and Technology, Japan
HIDEAKI HATA, Shinshu University, Japan
CHRISTOPH TREUDE, Singapore Management University, Singapore
KENICHI MATSUMOTO, Nara Institute of Science and Technology, Japan

GitHub's Copilot for Pull Requests (PRs) is a promising service aiming to automate various developer tasks related to PRs, such as generating summaries of changes or providing complete walkthroughs with links to the relevant code. As this innovative technology gains traction in the Open Source Software (OSS) community, it is crucial to examine its early adoption and its impact on the development process. Additionally, it offers a unique opportunity to observe how developers respond when they disagree with the generated content. In our study, we employ a mixed-methods approach, blending quantitative analysis with qualitative insights, to examine 18,256 PRs in which parts of the descriptions were crafted by generative AI. Our findings indicate that: (1) Copilot for PRs, though in its infancy, is seeing a marked uptick in adoption. (2) PRs enhanced by Copilot for PRs require less review time and have a higher likelihood of being merged. (3) Developers using Copilot for PRs often complement the automated descriptions with their manual input. These results offer valuable insights into the growing integration of generative AI in software development.

CCS Concepts: • **Social and professional topics → Software maintenance**; • **Software and its engineering → Automatic programming**.

Additional Key Words and Phrases: Pull Requests, Generative AI, Copilot, GitHub

ACM Reference Format:
Tao Xiao, Hideaki Hata, Christoph Treude, and Kenichi Matsumoto. 2024. Generative AI for Pull Request Descriptions: Adoption, Impact, and Developer Interventions. *Proc. ACM Softw. Eng.* 1, FSE, Article 47 (July 2024), 23 pages. https://doi.org/10.1145/3643773

1 INTRODUCTION

Generative AI is taking over many areas, and software development is no exception [Ebert and Louridas 2023]. As AI becomes more adept at creating content, code, and insights, developers are seeking innovative ways to work alongside these tools [Wang et al. 2020]. It is crucial to understand these interactions to gain insights into the evolution of AI in software development and how developers maintain the quality of software projects despite the AI's involvement.

Pull-based development [Gousios et al. 2016], due to its widespread use, has seen AI enhancements tailored for its processes. GitHub's Copilot for Pull Requests (PRs) is a prime example [GitHub Next 2023]. The tool is specifically designed to elevate the Pull Request experience: aiding developers in crafting optimised PR descriptions and enabling teams to review and merge PRs more efficiently.

Authors' addresses: Tao Xiao, Nara Institute of Science and Technology, Ikoma, Japan, tao.xiao.ts2@is.naist.jp; Hideaki Hata, Shinshu University, Nagano, Japan, hata@shinshu-u.ac.jp; Christoph Treude, Singapore Management University, Singapore, Singapore, ctreude@smu.edu.sg; Kenichi Matsumoto, Nara Institute of Science and Technology, Ikoma, Japan, matumoto@is.naist.jp.

ACM 2994-970X/2024/7-ART47
https://doi.org/10.1145/3643773

By tracking developers' work, suggesting tailored descriptions, and providing comprehensive code walkthroughs, it is transforming the review process. Copilot for Pull Requests allows developers to insert specific markers, which it then expands into rich content detailing the changes. For example, while `copilot:summary` gives a brief overview, `copilot:walkthrough` provides a detailed list of changes, including direct links to the relevant code. Thousands of PR descriptions have already benefitted from such automation.

With the tool's growing adoption, it is imperative to ask: How does it shape the code review process? Does it expedite reviews? Is there a higher chance of PRs being merged due to its input? Addressing these questions builds upon the existing research on factors influencing code review [Baysal et al. 2016], and the insights could be crucial for developers contemplating the adoption of generative AI tooling.

Copilot for PRs also offers a unique lens into human-AI collaboration in the context of software development [Wu et al. 2021]. With access to the full edit history of PRs, we can observe how developers adjust AI-generated content. Which details do they add or change? These interactions can reveal not only the dynamics between developers and AI tools but also hint at potential gaps and limitations of AI models in the software development arena.

To investigate these questions, we collected 18,256 PRs powered by Copilot for PRs from 146 GitHub projects, with 1,437 revisions that modified content suggested by Copilot for PRs, as well as 54,188 PRs that are not powered by Copilot for PRs from the same set of GitHub projects. This study is an exploratory study of early adoption of a new service during its limited release phase, similar to a previous study that examined early adopter usage when GitHub Discussions was in beta and only available to a limited number of users [Hata et al. 2022].

Our work answers the following research questions:

(RQ1) *To what extent do developers use Copilot for PRs in the code review process?*

(RQ2) *How are the code reviews affected by the use of Copilot for PRs?*

(RQ2.1) *Is there a relationship between the use of Copilot for PRs and review time?*

(RQ2.2) *Is there a relationship between the use of Copilot for PRs and the likelihood of a PR being merged?*

(RQ3) *How do developers adapt the content suggested by Copilot for PRs?*

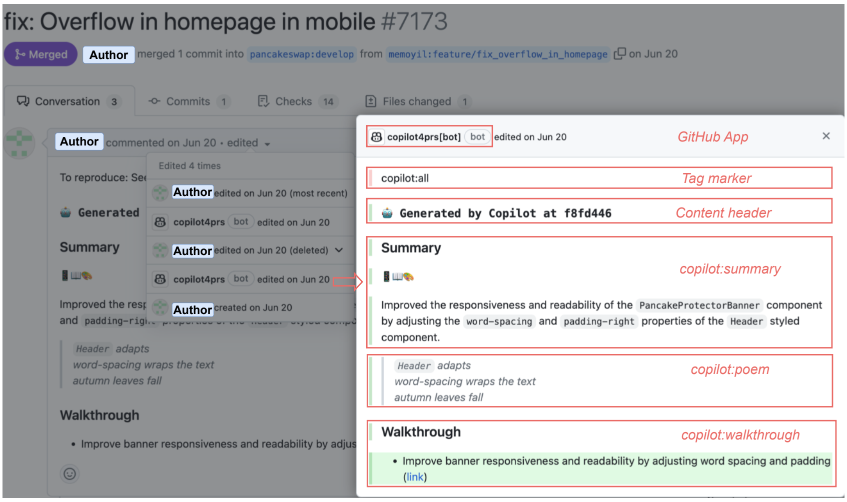
(RQ3.1) *What kind of supplementary information complements the content suggested by Copilot for PRs?*

(RQ3.2) *What kind of content suggested by Copilot for PRs undergoes subsequent editing by developers?*

We have observed (i) a burgeoning adoption of Copilot for PRs during code reviews. Repositories that have been using it for several months have extensively embraced this feature. Notably, `copilot:summary` stands out as the most popular marker tag, with 13,231 instances; (ii) the use of Copilot for PRs has resulted in a substantial reduction of review time by an average of 19.3 hours, with Pull Requests assisted by Copilot for PRs having a 1.57 times higher likelihood of being merged compared to those not assisted by Copilot for PRs; and (iii) there are 13 categories of supplementary information and seven editorial actions identified that developers employ on Copilot-suggested content. Developers often integrate templates (22.8%) and add relevant links (22.7%) to the content suggested by Copilot for PRs, while also frequently partially removing the suggested content (22.9%).

2 BACKGROUND

In this section, we provide an overview of Copilot for PRs (Section 2.1) and discuss its context within the broader literature concerning large language models and Pull Request summarization that is pertinent to this study (Section 2.2).



fix: Overflow in homepage in mobile #7173

Merged Author merged 1 commit into `pancakeswap:develop` from `memoyil:feature/fix_overflow_in_homepage` on Jun 20

Author commented on Jun 20 • edited

Edited 4 times

Author edited on Jun 20 (most recent)
copilot4prs bot edited on Jun 20
Author edited on Jun 20 (deleted)
copilot4prs bot edited on Jun 20
Author created on Jun 20

copilot4prs[bot] bot edited on Jun 20

copilot:all

Generated by Copilot at f8fd446

Summary

📱📖🎨

Improved the responsiveness and readability of the `PancakeProtectorBanner` component by adjusting the `word-spacing` and `padding-right` properties of the `Header` styled component.

> *`Header` adapts*
> *word-spacing wraps the text*
> *autumn leaves fall*

Walkthrough

- Improve banner responsiveness and readability by adjusting word spacing and padding (link)

Fig. 1. The GitHub Copilot for PRs feature: an example of `copilot:all`.

2.1 Copilot for PRs

With the increasing integration of Large Language Models (LLMs) into a variety of applications, GitHub has announced Copilot for Pull Requests, a tool designed to increase developer efficiency by integrating AI assistance into the code review process. The new service, which is not yet available to the public, extends GitHub's core Pull Requests (PRs) functionality by leveraging AI's ability to write PR descriptions and assist with the review and merge process [GitHub Next 2023]. Among the many features that Copilot for PRs is supposed to enable, there is a feature to generate pull request descriptions, only this feature allows developers to request to be waitlisted for use in a particular repository.[1] This feature allows developers to incorporate specific marker tags in their PR descriptions to append AI-generated content, courtesy of the `GPT-4` model by OpenAI. The marker tags and their corresponding functionalities are as follows:

- `copilot:summary`: Generates a concise summary of the changes encompassed in the PR.
- `copilot:walkthrough`: Provides a comprehensive list of modifications, each accompanied by links to the pertinent code segments.
- `copilot:poem`: Crafts a creative poem encapsulating the essence of the changes.
- `copilot:all`: Commands the inclusion of all the content types mentioned above.

We have found that several repositories have already been authorized to use and benefited from the feature of pull request description generation in Copilot for PRs. Figure 1 presents an example of the inclusion of the Copilot for PRs marker tag—`copilot:all`. In this instance, a GitHub App bot, specifically `copilot4prs`, automatically edits the PR description based on the provided marker tag. The bot generated a summary, a poem, and a walkthrough of the changes in this PR, as depicted in the figure. Developers can review the modified content, make additional adjustments, and even re-include a marker tag for subsequent commits.

2.2 Related Work

LLMs for SE. Since the introduction of the Transformer architecture in 2017 [Vaswani et al. 2017], LLMs have gained traction in the field of Software Engineering (SE). Hou et al. [2023] embarked on

[1] https://github.blog/2023-03-22-github-copilot-x-the-ai-powered-developer-experience/

a systematic review of 229 research articles focusing on the application of LLMs in SE from 2017 to 2023. The review revealed that a significant portion of the studies centered on solving problems in the software development domain. In this domain, LLMs, particularly GPT-2/GPT-3/GPT-3.5 [Dong et al. 2023; Li et al. 2023a; Liu et al. 2023a,b; Nascimento et al. 2023; Wang et al. 2023; Yetiştiren et al. 2023], GPT-4 [Bareiß et al. 2022; Gilbert et al. 2023; Jiang et al. 2023; Liu et al. 2023b], and the BERT series [Lai et al. 2023; Zeng et al. 2022], have shown a notable efficacy in tasks like code generation, code completion, and code summarization.

Code completion is an integral feature in Integrated Development Environments (IDEs) and code editors. Tools like Codex [Chen et al. 2021; Döderlein et al. 2022; Li et al. 2023b; Pearce et al. 2023], BERT series [Khan and Uddin 2022], GitHub Copilot [Döderlein et al. 2022; Li et al. 2023b; Pudari and Ernst 2023], CodeParrot [Li et al. 2023b; Xu et al. 2022], and GPT series [Ochs et al. 2023; Xu et al. 2022] offer intelligent and accurate code suggestions, greatly enhancing the coding process. In contrast, code summarization technologies like Codex [Ahmed et al. 2023; Arakelyan et al. 2023; Gao et al. 2023], CodeBERT [Chen et al. 2022; Gao et al. 2023; Gu et al. 2022], and T5 [Mastropaolo et al. 2022, 2021] focus on generating natural language descriptions from source code, promoting enhanced code maintenance, search, and classification.

Another application of LLMs in the software maintenance domain—accounting for nearly a quarter of the studies reviewed by Hou et al. [2023]—covers program repair, code review, and debugging. In the context of program repair, Codex [Wu et al. 2023; Xia et al. 2023] and ChatGPT [Xia and Zhang 2023] have exhibited strong performance. For code review tasks, LLMs like BERT [Sghaier and Sahraoui 2023] and ChatGPT [Sridhara et al. 2023] are instrumental in accurately detecting issues and suggesting code optimizations. Debugging, the process of identifying, locating, and resolving bugs, has also been enhanced by LLMs, with notable contributions from AutoSD [Kang et al. 2023] and SELF-DEBUGGING [Chen et al. 2023].

PR Summarization. PR summarization is closely related to Copilot for PRs. Liu et al. [2019] introduced Attn [Bahdanau et al. 2014], an attentional encoder-decoder model dubbed PRSummarizer, which leverages features from commit messages and added code comments to automatically generate PR descriptions. This model, assessed using the ROUGE metric [Lin 2004] and human evaluation, surpassed the performance of two baselines: LeadCM and LexRank [Erkan and Radev 2004]. Enhancing this work, Fang et al. [2022] proposed PRHAN, a novel hybrid attention network that effectively addresses the low efficiency and out-of-vocabulary (OOV) challenges identified in the model by Liu et al. [2019], delivering superior results.

Additionally, Irsan et al. [2022] delved into the realm of PR title generation, evaluating several summarization tools, including general-purpose models like BERTSumExt [Liu and Lapata 2019], BART [Lewis et al. 2019], and T5 [Raffel et al. 2020], as well as domain-specific models like PRSummarizer [Liu et al. 2019] and iTAPE [Chen et al. 2020]. Their assessments, based on the ROUGE metric and human evaluation, identified BART as the most effective tool for PR title generation. Their fine-tuned model, AUTOPRTITLE, which incorporates additional features, outperformed the others.

While previous research primarily concentrates on the development and evaluation of LLMs in code generation, completion, summarization, and maintenance tasks, our study is uniquely positioned in exploring the real-world applicability and impact of these models in a practical software development environment. We investigate Copilot for PRs, analyzing its impact on review time and merge decisions, and the edits made to the generated content. This holistic approach provides a comprehensive insight into the benefits and potential challenges associated with the integration of LLMs in everyday software engineering practices.

3 DATA COLLECTION

In this section, we outline our procedure for collecting data on PRs generated by Copilot for PRs (Section 3.1), PRs not generated by Copilot for PRs (Section 3.2), and the revisions of PRs generated by Copilot for PRs (Section 3.3).

3.1 PRs Generated by Copilot for PRs

To investigate Copilot's effectiveness in summarizing PRs, we began by compiling a list of PRs generated by Copilot for PRs. Utilizing GitHub GraphQL search,[2] we identified PRs containing the phrase "Generated by Copilot" in their descriptions, created on or before 31st August 2023. To manage the limitation of GitHub's maximum of 1000 responses per query, we implemented a strategy of dividing our search criteria (creation time in this case). Specifically, when a query for PRs within a certain time period yielded more than 1000 results, we halved the time period and repeated the search. This process was continued until the number of PRs fell within the 1000 result limit, ensuring we could efficiently gather all relevant data without exceeding GitHub's response constraints. We excluded false positives where developers included this phrase,[3] retaining only those PRs edited by `copilot4prs`, a GitHub bot for Copilot for PRs. This resulted in a total of 18,858 PRs across 150 GitHub repositories.

Identifying Obsolete Uses of Copilot for PRs. In our examination of PRs, we noted instances where developers tested Copilot for PRs on old PRs.[4] To maintain dataset integrity, we excluded PRs that were (i) closed before being edited by 'copilot4prs', and (ii) created before Copilot for PRs was introduced on GitHub (`2023-03-22 17:44:28+00:00`). This left us with 18,322 non-obsolete PRs.

Excluding PRs Submitted by Bots. We drew from the extensive study by Golzadeh et al. [2022] on bot detection techniques. Implementing two highly accurate methods, the "bot" suffix and the list of bots—based on 527 confirmed bots (e.g., `googlebot`) identified by Golzadeh et al. [[n. d.]]—we filtered out bot-submitted PRs. Moreover, including comments generated by bots could skew the representation of actual reviewer participation in code review discussions. To maintain the integrity and accuracy of our analysis, we focus exclusively on comments from human participants. Therefore we also removed comments generated by bots to enhance our dataset's quality. Consequently, we were left with 18,256 valid PRs from 146 GitHub repositories. The age of these 146 early adopters varies from 55 days (a forked repository for the purpose of exploring Copilot for PRs) to 4,762 days (`scikit-learn/scikit-learn`—a popular Python module for machine learning), with the average age of 1,360 days around four years.

3.2 PRs Not Generated by Copilot for PRs

To enable a comparative analysis, we also collated a dataset of PRs that were not generated by Copilot for PRs, sourced from the same 146 GitHub repositories. This approach ensures a balanced and fair comparison. Since review times for PRs can vary across different time periods, influenced by factors such as community growth, we selected PRs created from the introduction of the Copilot for PRs feature on GitHub to 31st August 2023 in each repository. This approach aims to control for temporal variations in review times. Like the Copilot-generated PRs, we applied a filtering criterion to exclude PRs submitted by bots to maintain consistency in data quality. This process yielded 54,188 PRs from 139 repositories for our comparative analysis (seven repositories were exclusively comprised of Copilot-generated PRs post-filtering during this specific timeframe). Table 1 presents

[2] https://docs.github.com/en/graphql/reference/queries#search
[3] For example, https://github.com/Kudoser/SteamTogether/pull/17
[4] https://github.com/argoproj/argo-cd/pull/1129

Table 1. The distribution of the state of studied PRs.

	Copilot-generated PRs		**non-Copilot-generated PRs**	
# merged	15,270	(84%)	38,639	(71%)
# closed	1,907	(10%)	12,056	(22%)
# opened	1,079	(6%)	3,493	(6%)
sum	18,256	(100%)	54,188	(100%)

the final count of PRs analyzed in this study. Notably, our initial observations indicate that the acceptance rate for Copilot-generated PRs (84%) is superior to that for non-Copilot-generated PRs (71%).

3.3 Revisions of PR Descriptions Generated by Copilot for PRs

We investigate how developers engage with the content suggested by Copilot for PRs (**RQ3**) by obtaining revisions of PR descriptions from 17,177 merged/closed PRs generated by Copilot for PRs.

Collection of Edits on PR Descriptions. We gather data on the edits made to each PR description, including information about who made the edit, the content changes, and the time of the edit. As noted in Section 2.1, `copilot4prs` serves as the editor, replacing marker tags with content suggested by Copilot for PRs. We collected 46,700 revisions from 17,177 merged/closed PRs generated by Copilot for PRs.

Exclusion of PRs without Post-Copilot Edits. Our focus is on understanding how developers adopt and adapt suggestions from Copilot for PRs during the code review process. Therefore, we exclude PRs that have not been edited following the initial placement of marker tags. This criteria yielded 18,486 revisions from 3,935 PRs.

Filtering PRs That Reapply Marker Tags. In our preliminary analysis of PRs generated by Copilot for PRs, we noticed instances where developers reapplied the same marker tag to examine the newly generated content after adding new commits. For instance, in this PR,[5] the author reapplied the `copilot:all` tag after adding commit #143ee5c. After applying this filter, we were left with 4,391 revisions from 730 PRs.

Identification of PRs with Post-Copilot Edits. We employ the `git diff` (Myers algorithm) on each content of revisions in PRs to trace modifications made to the content initially generated by `copilot4prs`. This process allowed us to identify 1,437 revisions spanning 311 PRs where developers made additional edits.

Identifying the PR Template. To segregate information inherent in the PR template, which is present prior to PR creation, we use the GitHub GraphQL API[6] to retrieve the PR template. We then pinpoint the version of the PR template closest to the PR creation time using `git log`. In cases where multiple PR templates could potentially auto-generate PR content, we calculate pairwise cosine similarities between each template and the initial PR description to identify the most similar template. These templates are subsequently used for answering **RQ3.1**.

4 METHODS

We present our mixed-methods procedure, comprising a quantitative analysis (**RQ1**), two quasi-experiments for causal inference (**RQ2**), and a qualitative analysis (**RQ3**).

[5] https://github.com/ultralytics/ultralytics/pull/1956

[6] https://docs.github.com/en/graphql/reference/objects#pullrequesttemplate

4.1 Quantitative Analysis

(RQ1) *To what extent do developers use Copilot for PRs in the code review process?*
To tackle **RQ1**, our quantitative examination of 18,256 valid PRs from 146 GitHub repositories emphasizes three elements related to the application of Copilot for PRs: (1) its adoption trajectory, (2) the extent of its employment in the code review mechanism, and (3) the dispersion patterns of Copilot for PRs marker tags.

Adoption Trend of Copilot for PRs. In order to understand the evolution in the acceptance of Copilot for PRs, we study the number of PRs that incorporated this service. Our exploration of its acceptance is limited to the initial phase of its existance, spanning from March 2023 to August 2023.

Proportions of Copilot for PRs. To assess the degree to which developers rely on this feature during code reviews, we compute the ratios of Copilot for PRs utilization, represented as $\frac{\#CopilotPRs}{\#PRs}$, within individual GitHub repositories using Copilot for PRs. Subsequently, a bubble plot is constructed to illustrate the temporal evolution of Copilot for PRs engagement across GitHub.

Distribution of Marker Tags from Copilot for PRs. Copilot for PRs introduces four unique marker tags[7] enabling customization of the generated content. Developers can incorporate these tags in their PR descriptions to elicit diverse responses from Copilot for PRs. To quantify the popularity of these marker tags, we use the regular expression for the extraction of their instances from the descriptions of PRs generated by Copilot for PRs.

```
\bcopilot:(all|summary|walkthrough|poem)\b
```

4.2 Casual Inference

(RQ2) *How are the code reviews affected by the use of Copilot for PRs?*

(RQ2.1) *Is there a relationship between the use of Copilot for PRs and review time?*

(RQ2.2) *Is there a relationship between the use of Copilot for PRs and the likelihood of a PR being merged?*

To answer these RQs, we build two quasi-experiments to estimate the causal impact of Copilot for PRs on reducing code review time in Pull Requests (**RQ2.1**) and increasing the likelihood of a Pull Request being merged (**RQ2.2**).

We define the review time as the time interval between the PR creation date and closed date in hours. In our analysis, we employ a statistical adjustment technique known as the Propensity Score Weighting (PSW) method to account for potential confounding factors in our observational data. This technique serves to calculate the inverse of the propensity score as a weight applied to each unit in the treatment group and the inverse of one minus the propensity score as a weight for each unit in the control group [Rubin 2001]. The propensity score itself is defined as the conditional probability of receiving treatment given a set of observed covariates [Rosenbaum and Rubin 1983]. Through this weighting scheme, the PSW method aims to construct a balanced pseudo-population in which the distribution of observed covariates is equivalent across both treatment and control groups [Rosenbaum and Rubin 1983; Rubin 2001]. In this analysis, the 17,177 (15,270 + 1,907) merged/closed PRs generated by Copilot for PRs as the treatment group, and the 50,695 (38,639 + 12,056) merged/closed PRs not generated by Copilot for PRs as the control group.

Explanatory Variables. Table 2 presents the 18 metrics that are used as explanatory variables in the logistic regression to estimate the propensity score for PSW. The treatment variable is whether Copilot for PRs is used to generate the PR description. The other 17 variables have been shown to have an impact on the review time, outcome, quality, and review participation of the modern code review process in previous studies [Kononenko et al. 2018; McIntosh et al. 2016; Thongtanunam et al. 2016, 2017; Tsay et al. 2014; Wang et al. 2021; Zhang et al. 2022]. Similar to

[7] https://web.archive.org/web/20231023053319/https://github.com/apps/copilot4prs

Table 2. The studied variables in **RQ2**.

PR variables	**Description**	**Median of Treatment**	**Median of Control**
# Added lines [Wang et al. 2021]	The number of added LOC by a PR.	**28**	25
# Deleted lines [Wang et al. 2021]	The number of deleted LOC by a PR.	**8**	7
PR size [Kononenko et al. 2018; McIntosh et al. 2016; Thongtanunam et al. 2016, 2017; Wang et al. 2021]	The total number of added and deleted LOC by a PR.	**44**	41
Purpose [McIntosh et al. 2016; Thongtanunam et al. 2017; Wang et al. 2021]	The purpose of a PR, i.e., bug, document, and feature.	-	-
# Files [Kononenko et al. 2018; Thongtanunam et al. 2017; Tsay et al. 2014; Wang et al. 2021]	The number of files changed by a PR.	**3**	2
# Commits [Kononenko et al. 2018; Tsay et al. 2014; Wang et al. 2021]	The number of commits involved in a PR.	**2**	**2**
Description length [McIntosh et al. 2016; Thongtanunam et al. 2017; Wang et al. 2021]	The length of a PR description.	**1,825**	343
PR author experience [Kononenko et al. 2018; Thongtanunam et al. 2017; Wang et al. 2021]	The number of prior PRs that were submitted by the PR author.	**155**	151
Is member [Tsay et al. 2014; Zhang et al. 2022]	Whether or not the PR author is a member or outside collaborator (binary).	-	-
# Comments [Kononenko et al. 2018; Wang et al. 2021]	The number of comments left on a PR.	0	**1**
# Author comments [Kononenko et al. 2018; Wang et al. 2021]	The number of comments left by the PR author.	**0**	**0**
# Reviewer comments [Kononenko et al. 2018; Wang et al. 2021]	The number of comments left by the reviewers who participate in the discussion.	**0**	**0**
# Reviewers [Thongtanunam et al. 2016; Wang et al. 2021]	The number of developers who participate in the discussion.	**0**	**0**
Repo age [Tsay et al. 2014; Zhang et al. 2022]	Time interval between the repository creation time and PR creation time in days.	1,060	**1,250**
Project variables			
Language [Zhang et al. 2022]	The repository language that a PR belongs to, represented by the top 10 or others.	-	-
# Forks [Zhang et al. 2022]	The number of forks that a repository has.	264	**286**
# Stargazers [Tsay et al. 2014; Zhang et al. 2022]	The number of stargazers that a repository has.	1,359	**1,654**
Treatment variables			
With Copilot for PRs	Whether or not a PR is generated by Copilot for PRs (binary).	-	-
Outcome variables			
Review time **(RQ2.1)**	Time interval between the PR creation time and closed time in hours.	12.17	**16.09**
Is merged **(RQ2.2)**	Whether or not a PR is merged (binary).	-	-

the prior work [McIntosh et al. 2016; Mockus and Votta 2000; Thongtanunam et al. 2017; Wang et al. 2021], we classify the purpose of a PR for which the description contains “doc”, “copyright” or “license” as documentation, and if a PR description contains “fix”, “bug”, or “defect”, it is classified as bug fixing. The remaining PRs are classified as feature introduction. We adopt the fork count and stargazer count as project variables since they are indications of the attention a project receives. Moreover, different primary programming languages of projects may also infer the review time. Additionally, we identified whether or not the PR author is a member or outside collaborator according to the GitHub GraphQL.[8] We count ‘MEMBER’ and ‘OWNER’ as members of a GitHub repository.

Entropy Balancing. We employed the Entropy Balancing method [Hainmueller 2012], which ensures optimal balance for specified statistical moments of the covariates while minimizing the entropy of the weights. The reasoning behind this choice is that Entropy Balancing offers a more flexible and efficient way to reweight units while retaining valuable information in the preprocessed data, preventing the need for continual balance checking and iterative searching over propensity score models, which may fail to balance covariate distributions in finite samples [Hainmueller 2012]. Figure 2 illustrates the reduction in absolute mean differences as a result of applying this method, transitioning from unadjusted to adjusted states (where ‘unadjusted’ encompasses all PRs prior to balancing, and ‘adjusted’ refers to those post-balancing). None of the absolute mean differences of adjusted exceeds 0.10, which means that we obtained weighted variables for the treatment and control groups with a balanced distribution of covariates.

Treatment and Outcome Variable. To estimate the impact of Copilot for PRs usage in code reviews, a linear regression is performed using the above variables and the variable `treatment`, which takes a value of 0 or 1 that indicates the presence or absence of Copilot for PRs. On the one

[8] https://docs.github.com/en/graphql/reference/enums#commentauthorassociation

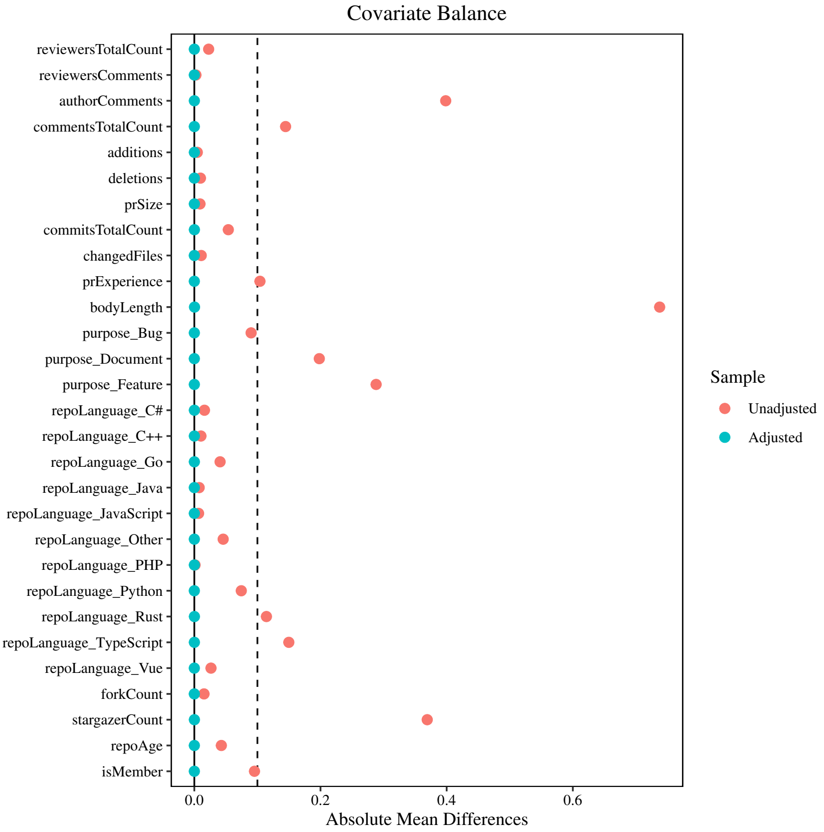


Fig. 2. Covariate balance before (unadjusted) and after (adjusted) propensity score weighting.

hand, the outcome variable for **RQ2.1** is the review time, defined as the time interval between the PR creation date and the date it was closed, measured in hours. On the other hand, the outcome variable for **RQ2.2** is whether or not a PR is merged, where we estimate the causal impact of the use of Copilot for PRs on the likelihood of a PR being merged.

4.3 Qualitative Analysis

(RQ3) *How do developers adapt the content suggested by Copilot for PRs?*
For the qualitative analysis examining how developers complement and modify content suggested by Copilot for PRs, we qualitatively analysed 1,437 revisions from 311 PRs where developers made additional edits as illustrated in Section 3.3. To increase confidence in our qualitative processes, three of the authors initially collaboratively reviewed a set of 30 samples. This preliminary examination focused on identifying recurring themes relevant to our research questions. Following this, one author formalized these discussions into a well-defined coding schema. Subsequently, three authors employed this coding schema to analyze another 30 samples. One author then finalized the annotations based on the encouraging kappa agreements. We allowed multiple codes per PR. In all cases, the kappa agreement proved satisfactory on the first pass, obviating the need for further revisions.

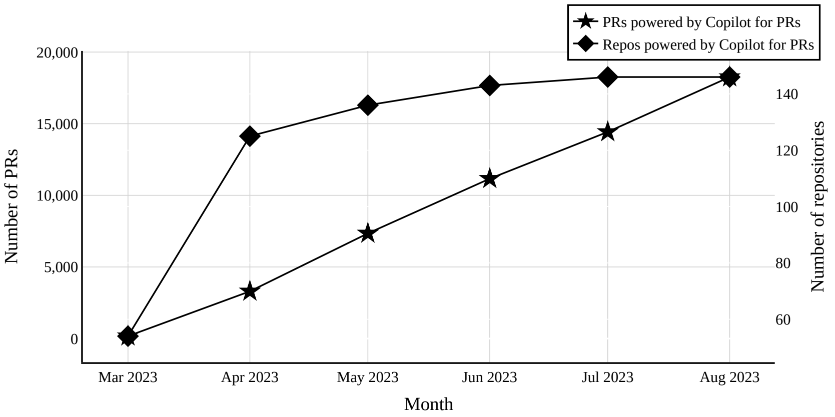


Fig. 3. Cumulative time-series of PRs using Copilot for PRs vs. Non-Copilot for PRs.

(RQ3.1) *What kind of supplementary information complements the content suggested by Copilot for PRs?* To answer **RQ3.1**, we analyzed how developers complement the content suggested by Copilot for PRs. Three raters independently coded 30 samples, achieving a kappa of 0.64 or ‘substantial’ agreement [Viera et al. 2005]. The lower agreement can be explained by 24 combinations of multiple codes being discovered when coding complementary information to the content suggested by Copilot for PRs. Three raters achieved perfect agreement in 15/30 cases (50%), partial agreement in another 14/30 cases (47%), and completely disagreed only in 1/30 case (3%). Our coding schema and the frequencies of different codes are shown in Table 5.

(RQ3.2) *What kind of content suggested by Copilot for PRs undergoes subsequent editing by developers?* To answer **RQ3.2**, we analyzed how developers modified the content suggested by Copilot for PRs. Three raters independently coded 30 samples, achieving a kappa of 0.62 or ‘substantial’ agreement [Viera et al. 2005]. The lower agreement can be explained by 26 combinations of multiple codes being discovered when coding modified information of the content suggested by Copilot for PRs. Three raters achieved perfect agreement in 14/30 cases (47%), partial agreement in another 15/30 cases (50%), and completely disagreed only in 1/30 case (3%). Our coding schema and the frequency of different codes are shown in Table 6.

5 RESULTS

In this section, we present the results of our research questions 1–3.

5.1 RQ1: To what extent do developers use Copilot for PRs in the code review process?

Figures 3–4 and Table 3 show the results of our analysis. We now discuss our results below.

Adoption Trend of Copilot for PRs. Figure 3 presents the cumulative time-series of PRs using Copilot for PRs. We began collecting PRs created after the initial instance of a PR in GitHub featuring Copilot for PRs, which occurred on `2023-03-22 17:44:28+00:00`. The number of repositories using Copilot for PRs has barely increased since the first month. It seems that only a limited number of repositories are accepted as early adopters of this feature. On the other hand, the number of pull requests containing automatically generated descriptions by Copilot for PRs has been steadily increasing. This suggests that early adopters are actively using this new feature.

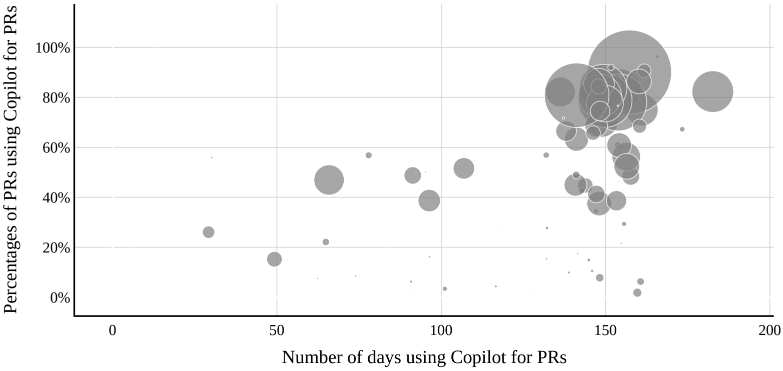


Fig. 4. Proportions of PRs using Copilot for PRs per repository.

Proportions of Copilot for PRs. Figure 4 depicts the bubble plot of the proportions of PRs using Copilot for PRs across repositories. In the bubble plot, the y-axis represents the rates of PRs utilizing Copilot for PRs, while the size of each bubble corresponds to the number of PRs in that repository using Copilot for PRs. The x-axis showcases the duration for which Copilot for PRs has been employed in the repository. From the bubble plot, we can see that repositories with long experience periods (x-axis) with Copilot for PRs have many PRs with Copilot for PRs (bubble size is bigger) and its ratio (y-axis) is also high. Notably, 50 repositories have a heavy reliance on Copilot for PRs, with over 50% of their PRs being powered by it. In more extreme cases, seven repositories have wholly incorporated Copilot for PRs into their PR descriptions or their PR description templates, a topic further explored in **RQ3**. Conversely, 96 repositories only sporadically employ this feature, i.e., rates less than or equal to 50% of PRs.

Distribution of Marker Tags in Copilot for PRs. We obtained a total of 31,379 instances of marker tags from 18,256 PRs generated by Copilot for PRs. The top portion of Table 3 shows the distribution of marker tags in PRs generated by Copilot for PRs, while the bottom portion of Table 3 presents the combinations of marker tags in these PRs. We find that the `copilot:summary` tag is the most frequently occurring, accounting for 13,231 instances, while `copilot:poem` emerges as the least frequent. The four most common combinations of marker tags are associated with Summary and Walkthrough content (5,598 PRs), only the All content (4,512 PRs), only the Summary content (3,725 PRs), and a combination of Summary, Poem, and Walkthrough content (2,772 PRs). We discovered 14 PRs with no marker tags in their descriptions, as these commented-out marker tags (we identified marker tags by the commented-out contents in PR descriptions) were removed by developers.

RQ1 Summary: We observe the growing adoption of Copilot for PRs in the code review process. While a substantial portion of repositories have adopted this feature extensively, over half utilize it only minimally. The most commonly employed Copilot for PRs marker tag is `copilot:summary`, with 13,231 instances. The most popular combination is associated with `copilot:summary` and `copilot:walkthrough`, accounting for 5,598 PRs.

Table 3. Distribution of marker tags in PRs generated by Copilot for PRs.

	Category	#
Marker Tag	copilot:summary	13,231
	copilot:walkthrough	8,990
	copilot:all	4,952
	copilot:poem	4,206
Combinations of Marker Tags	copilot:summary and copilot:walkthrough	5,598
	copilot:all	4,512
	copilot:summary	3,725
	copilot:summary, copilot:poem, and copilot:walkthrough	2,772
	copilot:summary and copilot:poem	716
	copilot:all, copilot:summary, copilot:poem, and copilot:walkthrough	412
	copilot:poem	275
	copilot:walkthrough	195
	copilot:all and copilot:poem	19
	none	14
	copilot:poem and copilot:walkthrough	9
	copilot:all, copilot:summary, and copilot:poem	3
	copilot:all, copilot:summary, and copilot:walkthrough	3
	copilot:all and copilot:summary	2
	copilot:all and copilot:walkthrough	1

5.2 RQ2: How are the code reviews affected by the use of Copilot for PRs?

(RQ2.1) *Is there a relationship between the use of Copilot for PRs and review time?* Table 2 presents the studied variables and their median values in the treatment group and control group. We observe that the median value of outcome variables (i.e., review time) of the treatment group (i.e., 12.17 hours) is less than the control group (i.e., 16.09 hours). Table 4 summarizes the regression result. As seen in the coefficient estimate of `treatment`, there is a statistically significant positive effect of Copilot for PRs on reducing code review time in RPs. As the average of the expected causal effect of treatment on individuals in the treatment group, called Average Treatment Effects on the Treated (ATT), we find that Copilot for PRs has an impact of decreasing the review time by 19.3 hours.

(RQ2.2) *Is there a relationship between the use of Copilot for PRs and the likelihood of a PR being merged?* To compute the marginal log odd ratio (OR) for a PR being merged, we employ `avg_comparisons()` with the compassion of `lnoravg` for the binary treatment variable [Austin 2022; Austin and Stuart 2017].[9] We observe that estimated odds ratio for the treatment variable (with Copilot for PRs) is 1.57, with a 95% confidence interval ranging from 1.35 to 1.84. The `p-value` associated with this estimate was less than 0.001, indicating statistical significance at conventional alpha levels. This result implies that Pull Requests generated with the aid of Copilot for PRs are approximately 1.57 times more likely to be merged than those created without it. Given the narrow confidence interval and the statistical significance of the estimate, we can confidently assert that the treatment variable (with Copilot for PRs) exerts a meaningful impact on the likelihood of a Pull Request being merged.

[9] https://ngreifer.github.io/WeightIt/articles/estimating-effects.html#binary-outcomes

Table 4. Summaries of causal inference estimating the effect of Copilot for PRs.

	estimate	std. error	p
treatment	**-19.3**	2.27	**1.64e-17**
reviewersTotalCount	15.0	1.59	4.17e-21
reviewersComments	0.996	0.836	2.34e-1
authorComments	38.8	1.26	3.21e-206
commentsTotalCount	NA	NA	NA
additions	0.0000266	0.0000198	1.79e-1
deletions	-0.000000194	0.0000413	9.96e-1
prSize	NA	NA	NA
commitsTotalCount	0.753	0.0431	4.03e-68
changedFiles	-0.0167	0.00879	5.76e-2
prExperience	-0.0503	0.00288	4.40e-68
bodyLength	0.00261	0.0000802	8.79e-231
purposeDocument	2.12	2.59	4.12e-1
purposeFeature	-0.623	2.89	8.30e-1
repoLanguageC++	-49.5	8.71	1.32e-8
repoLanguageGo	-66.2	6.16	6.36e-27
repoLanguageJava	-46.2	7.99	7.30e-9
repoLanguageJavaScript	-64.5	6.85	4.79e-21
repoLanguageOther	-52.3	6.23	5.24e-17
repoLanguagePHP	-9.75	9.02	2.80e-1
repoLanguagePython	21.0	6.64	1.58e-3
repoLanguageRust	-34.3	7.35	3.21e-6
repoLanguageTypeScript	-53.5	5.56	7.66e-22
repoLanguageVue	-44.9	7.81	9.01e-9
forkCount	-0.000364	0.000276	1.87e-1
stargazerCount	-0.0000786	0.000118	5.05e-1
repoAge	0.0190	0.00118	1.16e-58
isMember	-28.5	2.27	3.85e-36

NA indicates the Perfect Collinearity, where this variable can be predicted by other variables.

RQ2 Summary: Using Copilot for PRs has a positive impact on reducing review time, reducing it by 19.3 hours. It also has a positive impact on the likelihood of a Pull Request being merged, with a 1.57 times higher chance of being merged compared to PRs whose descriptions were not generated by Copilot for PRs.

5.3 RQ3: How do developers adapt the content suggested by Copilot for PRs?

From 13 categories of complementary information in Table 5 and seven main categories of editorial actions in Table 6, we observe that the common category for complementary information is static template information, comprising 22.8% of the total. Closely following is the associated link at 22.7%. Meanwhile, both the execution log and performance impact are notably less common, each making up 0.7%. In terms of editorial actions, deletions emerge as the most common at 22.9%, with

Table 5. Definition and frequency of complementary information categories.

Category	Definition	Frequency
Static Template Information	Pre-existing content that resides in the PR template and does not require modification, except for checklists.	137 (22.8%)
Associated Link	Reference link or identifier corresponding to a related issue, Pull Request, or documentation.	136 (22.7%)
Pull Request Intent	A brief description outlining the objectives or intention of the PR.	77 (12.8%)
Testing Information	Procedures and results of testing.	55 (9.2%)
Custom Changelog	A developer-defined changelog.	46 (7.7%)
Visual Representation	Graphical elements (e.g., image) that provide visual context (e.g., diagram, testing screenshot, change screenshot) for the changes.	44 (7.3%)
Future Tasks	A list of tasks or objectives to be completed in the future.	13 (2.2%)
Code Snippet	Code excerpts that illustrate specific changes implemented in the PR or test scripts.	8 (1.3%)
Reproduction Steps	Detailed step-by-step guide on replication of the observed behavior or issue.	5 (0.8%)
Authorship	Certification to prove the contributor has signed off on the contribution.	5 (0.8%)
Execution Log	Compiled log files representing the outcome of code execution or test runs.	4 (0.7%)
Performance Impact	Impact of the PR on the software's performance metrics.	4 (0.7%)
No Information	No supplementary information is provided.	66 (11.0%)

refinement and exclusion also being significant at no less than 17.4%. Augmentation, on the other hand, is relatively infrequent at 6%. In the following, we show examples of the most common and worth-noting categories.

(RQ3.1) *What kind of supplementary information complements the content suggested by Copilot for PRs?* The most prevalent form of additional information (accounting for 22.8%) is static template information. While these PR templates are not our central concern, we advocate for a more in-depth exploration of this facet, as detailed by Li et al. [2022]. We also found some GitHub repositories used the Copilot for PRs marker tags in their Pull Request template.[10] The second most frequent category of additional content is associated links, which serve to reference related software artifacts. We provide the example[11] below for both static template information and associated links.

```
Fixes #10647
<details>
Copilot Summary
</details>

## Checklist for CI (.github/workflows) changes
- [ ] If changed package build workflow, pass [this
↪  action](https://github.com/emqx/emqx/actions/workflows/build_packages.yaml)
↪  (manual trigger)
- [ ] Change log has been added to `changes/` dir for user-facing artifacts update
```

Developers sometimes augment the PR descriptions with contextual information that Copilot for PRs fails to capture, such as the intent behind the PR, testing, and images. Notably, some developers

[10] https://github.com/vlang/v/blob/master/.github/PULL_REQUEST_TEMPLATE#L47
[11] https://github.com/emqx/emqx/pull/11276

even maintain their own changelogs within PRs, accounting for 7.7% of the PRs. In the example[12] below, the developer listed the changes to complement the content suggested by Copilot for PRs.

```
- Lifts gdoc state out of the preview page component and puts it into the `GdocsStore`
- Restyles index page
- Adds search

<details>
Copilot Summary
</details>
```

(RQ3.2) *What kind of content suggested by Copilot for PRs undergoes subsequent editing by developers?* Deletion emerges as the most frequent editorial action (22.9%), which indicates developers partially choose the Copilot-generated content based on their needs. For instance,[13] one developer removed superfluous PR improvement statements from a Copilot-generated summary (text with a strikethrough indicating content that has been deleted).

> 🤖 Generated by Copilot at 7a46a2f
> Reduced the log level of some messages in util.cpp to avoid cluttering the output with non-critical information. ~~This improves the readability and performance of the lane change planner.~~

It is worth noting that some PR authors opt to eliminate all links to code diffs, presumably to maintain the aesthetic integrity of the PR. In the example[14] below, the developer removed both the link references and a change item represented as a bullet point in the Walkthrough section.

> 🤖 Generated by Copilot at 564357f
> - Add a production debug feature that allows the user to enable the dev utils button by tapping seven times on the commit hash text in the settings screens ~~(link, link, link, link, link, link)~~
> - ~~Add a CopyLogsButton component that allows the user to copy the application logs to the clipboard (link, link)~~

Additionally, developers often fine-tune the phrasing or terminology used by Copilot for PRs, such as correcting variable names. Below, we provide an example[15] where the variable name ‘BASIC_METADATA_OVERRIDE_KEYS’ was corrected to ‘BASIC_METADATA_KEYS’ (text with a strikethrough indicating content that has been deleted, while bold text indicating content that has been added).

> 🤖 Generated by Copilot at 3454d39
> This pull request adds a new feature that allows customizing the file asset metadata fields that are exposed by the API. It introduces a new configuration property ~~‘BASIC_METADATA_OVERRIDE_KEYS’~~**‘BASIC_METADATA_KEYS’** and modifies the ‘FileMetadataAPIImpl’ class to use it.

[12] https://github.com/owid/owid-grapher/pull/2213
[13] https://github.com/autowarefoundation/autoware.universe/pull/3369
[14] https://github.com/trezor/trezor-suite/pull/7962
[15] https://github.com/dotCMS/core/pull/25745

Table 6. Definition and frequency of editorial action categories.

Category	Definition	Frequency	
Deletion	Eliminate the Copilot-generated content by majorly (i.e., at least two sentences) or partially (i.e., up to one sentence). Additionally, the developer sometimes eliminates any referenced URLs in the Walkthrough or drops the header generated by Copilot for PRs.	**95**	**22.9%**
Partial Summary Deletion		33	8.0%
Eliminate Walkthrough Bullet Point		27	6.5%
Major Summary Deletion		14	3.4%
Remove Copilot Header		9	2.2%
Partial Walkthrough Deletion		5	1.2%
Omit Diff Links in Walkthrough		4	1.0%
Major Walkthrough Deletion		1	0.2%
Major Poem Deletion		1	0.2%
Partial Poem Deletion		1	0.2%
Refinement	Refine the Copilot-generated content by majorly (i.e., at least two sentences) or minorly (i.e., up to one sentence).	**86**	**20.8%**
Minor Summary Refinement		48	11.6%
Minor Walkthrough Refinement		17	4.1%
Major Summary Refinement		16	3.9%
Minor Poem Refinement		3	0.7%
Major Walkthrough Refinement		2	0.5%
Exclusion	Exclude the entire Copilot-generated content, the extraneous characters added before and after marker tags by developers, or the duplicate content generated by Copilot for PRs.	**72**	**17.4%**
Exclude Poem		34	8.2%
Exclude Walkthrough		22	5.3%
Exclude Summary		10	2.4%
Remove Developer-Added Extraneous Characters		5	1.2%
Remove Duplicate Copilot Content		1	0.2%
Replacement	Substitute the Copilot-generated content (including the header or emoji) to a developer-defined content.	**61**	**14.7%**
Replace Summary		46	11.1%
Replace Copilot Header		11	2.7%
Replace Walkthrough		3	0.7%
Replace Link in Walkthrough		1	0.2%
Exchangement	Switch/rearrange the Copilot-generated content by the need.	**51**	**12.3%**
Rearrange Copilot-Generated Content		18	4.3%
Switch Summary to Comprehensive Content		13	3.1%
Switch Walkthrough to Comprehensive Content		10	2.4%
Switch Comprehensive Content to Summary		3	0.7%
Switch Walkthrough to Summary		2	0.5%
Switch Summary to Poem		2	0.5%
Switch Summary to Walkthrough		1	0.2%
Switch Poem to Walkthrough		1	0.2%
Switch Poem to Comprehensive Content		1	0.2%
Augmentation	Incorporate additional changes that Copilot for PRs failed to include in the Copilot-generated content, and context information, e.g., the PR impact, PR intent, and explanation for Copilot for PRs emoji.	**25**	**6.0%**
Augment Summary		19	4.6%
Augment Walkthrough		4	1.0%
Add explanations to Copilot emoji		2	0.5%
False Positive	Incorrectly categorized as edited.	**24**	**5.8%**

Another example[16] of a refinement is shown in the following. The developer rephrased a term to describe changes in a PR (i.e., 'unused methods' to 'duplicate method definitions').

> 🤖 Generated by Copilot at 8a75e3a
> This Pull Request removes unnecessary quotes around type annotations in various files and classes, following the PEP 484 style guide for type hints. This improves the readability and consistency of the code and avoids the need for forward references. It also removes some ~~unused methods~~**duplicate method definitions** from the WandbCallback class and simplifies the string formatting in the results_data_frame function.

Moreover, developers prune extraneous material from the `copilot:all` tag, with 11% specifically removing `copilot:poem`. They also replace Copilot-generated content with their own (14.7%) or exchange Copilot-generated content for another type of Copilot-generated content (12.3%). As shown in the example[17] below, the developer commented on the summary by reformatting it as strikethrough to abort this content suggested by Copilot for PRs.

> 🤖 Generated by Copilot at b807adc
> ~~This Pull Request adds support for the Skia backend to the SamplesApp project, refactors the rendering and extension logic of the Uno.UI.Composition and Uno.UI.Runtime.Skia.Gtk projects, and aligns the app manifest of the SamplesApp.UWP project with the Uno Platform branding. It also renames, deletes, or updates some files and namespaces to improve the code organization and consistency.~~
> Nope, you didn't get it this time

In 6% of instances, developers add additional changes or context information to augment the content suggested by Copilot for PRs. In the following example,[18] the developer added hidden emoji explanations (which were commented out in the PR description) and replaced the summary with a developer-defined summary.

> 🐛📊🌈 **<– copilot thought of this (and also gave an explanation, see below)**
>
> ~~Fix highlighting bugs in StackedAreaChart and StackedBarChart when using external legends. Use rawSeries instead of series to match the legend names with the chart data.~~
> **Fixes a problem in StackedArea and StackedBar charts where hovering over an entity in the legend didn't grey out all other entities. Here, Italy is hovered over:**
> **Emoji legend (generated by copilot):**
> 🐛 **- This emoji represents a bug fix, which is the main purpose of these changes. The bug was causing the wrong series to be highlighted on the stacked area and bar charts when using an external legend.**
> 📊 **- This emoji represents a chart or graph, which is the type of component that these changes affect. The stacked area and bar charts are both chart components that use an external legend to display the series names and colors.**

[16] https://github.com/wandb/wandb/pull/5940
[17] https://github.com/unoplatform/uno/pull/12596
[18] https://github.com/owid/owid-grapher/pull/2109

> **🌈 - This emoji represents a rainbow or color, which is a relevant aspect of these changes. The highlighting feature of the external legend and the chart components depends on the color of the series, and these changes ensure that the correct color is used for the correct series.**

Lastly, we identified 24 false positives in our dataset, occurring when Copilot for PRs and developers edited the PR description simultaneously, leading to content reappearances,[19] or when the developer deleted the edit history,[20] or when Copilot for PRs restored the edited PR template to its original state.[21]

> **RQ3 Summary:** We identified 13 categories of supplementary information and observed seven edit actions to content suggested by Copilot for PRs. For strategies of adapting AI generated content, developers tend to include templates and associated links to complement it. Moreover, they often partially removed content suggested by Copilot for PRs.

6 DISCUSSION

We now discuss the recommendations from our results, as well as the threats to the validity of our study.

6.1 Recommendations

Based on our findings, we make the following recommendations for practitioners, researchers, and GitHub Copilot for PRs. First, we recommend that practitioners:

- *Advocate for Copilot for PRs adoption in PRs:* According to our results (**RQ2**), PRs enriched by Copilot for PRs generally necessitate less review time and exhibit an increased likelihood of being merged. We champion the utilization of AI-powered descriptions to amplify the clarity of the changes outlined in PRs.
- *Incorporate Copilot for PRs tags into PR templates:* Our data reveals occasional developer inclusion of Copilot for PRs tags in PR templates. Additionally, templates often act as a supplemental layer to Copilot-generated content (**RQ3.1**). Thus, we advise practitioners to harmonize their unique PR requirements with these AI-generated tags.
- *Exercise discretion with the `copilot:all` tag:* Our **RQ3.2** results indicate that developers frequently exclude the ‘poem’ section when employing the `copilot:all` tag. To minimize confusion for reviewers, it is advisable to be selective when including various types of Copilot-generated content.

For researchers, we suggest:

- *Examine content evolution across commits:* As noted in Section 3.3, developers often redeploy Copilot for PRs tags to assess generated content against new commits. A nuanced qualitative analysis of such behavior poses a prospective research direction.
- *Establish Copilot for PRs tag integration guidelines:* Although we advocate for the incorporation of Copilot for PRs tags into PR templates, devising guidelines based on the specific categories behind PR templates [Li et al. 2022] could prove beneficial.

Future research directions that could augment GitHub Copilot for PRs include:

[19] https://github.com/dotCMS/core/pull/25770
[20] https://github.com/pancakeswap/pancake-frontend/pull/7173
[21] https://github.com/lensterxyz/lenster/pull/2413

- *Refinement of template integration:* Given that templates frequently complement Copilot-generated content (**RQ3**), Copilot for PRs could benefit from offering more customized or comprehensive content tailored to specific repositories.
- *Offer developer-specific suggestions:* Our findings in **RQ3.2** indicate a propensity for developers to tailor suggestions from Copilot for PRs. Allowing for such customization could further enhance the utility of Copilot for PRs.
- *Learn from Developer Modifications:* As observed in **RQ3.2**, developers frequently modify, remove, or augment specific elements within suggestions from Copilot for PRs. Capturing these alterations could provide valuable learning opportunities for enhancing the platform's generated content.

6.2 Threats to Validity

Below, we outline potential threats to the validity of our study:

Construct Validity: This study focuses on how developers adopt Copilot for PRs, thereby excluding PRs submitted by bots. We identify bots based on a "bot" suffix and employ techniques by Golzadeh et al. [2022]. Although these methods are highly precise, they might not capture every true negative, potentially influencing the construct validity.

Internal Validity: Our study hinges on manually coded data, which is susceptible to miscoding due to the subjective interpretations of the coding schema. To counteract this, we adhere to two best practices for open coding: 1) we execute two rounds of independent coding and compute the Kappa statistic to assure a 'Substantial' level of agreement, and 2) if the coding schema undergoes modifications, we revisit and adjust the existing coding accordingly. Additionally, we construct our quasi-experiments considering 17 confounding variables (as detailed in Table 2). These variables have been shown to have an influence on review time, outcome, quality, and participation. However, there may be other confounding variables not accounted for, necessitating further analysis. Furthermore, we assessed the balance achieved by propensity score weighting, focusing on quality assessment through this method rather than traditional fit statistics like R-squared. This approach aligns with the perspective that traditional goodness-of-fit tests are of limited relevance in this context. The primary goal of weighting is to balance covariate distributions within the sample, rather than to infer assignment probabilities in the overall population [Li et al. 2018].

External Validity: GitHub Copilot for PRs was introduced in March 2023, making limited users early adopters. Given the limited number of developers who have access at this stage, there is an inherent threat to the external validity of our empirical findings. As such, it is crucial to clarify that our results are not universally applicable to the broader open-source developer community, but are more pertinent to these early adopters. Developers who are less eager to adopt new technologies might use Copilot for PRs less or differently compared to the early adopters studied in this work.

7 CONCLUSION

In this work, we examined how developers are using generative AI for writing and reviewing Pull Requests (PRs) and its effects on the code review process. Our study included over 18,000 PRs assisted by Copilot for PRs and more than 54,000 that were not. We found a growing trend of Copilot for PRs use in code reviews. Some repositories have fully embraced it, while others are still testing the waters.

Our data shows that Copilot for PRs can reduce the time needed for code reviews and increase the chances of PRs getting merged. Developers are also augmenting the AI-generated content, demonstrating a unique interplay where human expertise edits and refining the machine-generated suggestions to ensure contextual relevance and technical accuracy.

Looking ahead, our exploration into generative AI for PR descriptions is just the beginning of exploring the potential for human-AI collaboration in the context of code reviews and other software development tasks. We envision future work on empirically exploring the adoption of AI and the adaptive strategies employed by developers to tailor AI outputs in the areas of code reviews, code creation, and documentation, among others.

DATA AVAILABILITY

Our replication package [Xiao et al. 2024] includes lists of studied PRs from GitHub, both with and without the use of Copilot for PRs. It also provides the features of PRs that were either generated or not generated by Copilot for PRs (pertaining to RQ2), coding results for RQ3, and scripts. The complete replication package can be also accessed at https://github.com/NAIST-SE/CopilotForPRsEarlyAdoption.

ACKNOWLEDGMENTS

This work was supported by JSPS Grant-in-Aid for JSPS Fellows JP23KJ1589, JSPS KAKENHI Grant Numbers JP20H05706, and JST PRESTO Grant Number JPMJPR22P6.

REFERENCES

Toufique Ahmed, Kunal Suresh Pai, Premkumar Devanbu, and Earl T Barr. 2023. Improving Few-Shot Prompts with Relevant Static Analysis Products. *arXiv preprint arXiv:2304.06815* (2023).

Shushan Arakelyan, Rocktim Jyoti Das, Yi Mao, and Xiang Ren. 2023. Exploring Distributional Shifts in Large Language Models for Code Analysis. *arXiv preprint arXiv:2303.09128* (2023).

Peter C Austin. 2022. Bootstrap vs asymptotic variance estimation when using propensity score weighting with continuous and binary outcomes. *Statistics in Medicine* 22 (2022), 4426–4443. https://doi.org/10.1002/sim.9519

Peter C Austin and Elizabeth A Stuart. 2017. Estimating the effect of treatment on binary outcomes using full matching on the propensity score. *Statistical methods in medical research* 6 (2017), 2505–2525. https://doi.org/10.1177/0962280215601134

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

Patrick Bareiß, Beatriz Souza, Marcelo d'Amorim, and Michael Pradel. 2022. Code generation tools (almost) for free? a study of few-shot, pre-trained language models on code. *arXiv preprint arXiv:2206.01335* (2022).

Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. 2016. Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering* (2016), 932–959. https://doi.org/10.1007/s10664-015-9366-8

Fuxiang Chen, Fatemeh H Fard, David Lo, and Timofey Bryksin. 2022. On the transferability of pre-trained language models for low-resource programming languages. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 401–412. https://doi.org/10.1145/3524610.3527917

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2020. Stay professional and efficient: automatically generate titles for your bug reports. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 385–397. https://doi.org/10.1145/3324884.3416538

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128* (2023).

Jean-Baptiste Döderlein, Mathieu Acher, Djamel Eddine Khelladi, and Benoit Combemale. 2022. Piloting Copilot and Codex: Hot Temperature, Cold Prompts, or Black Magic? *arXiv preprint arXiv:2210.14699* (2022).

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration Code Generation via ChatGPT. *arXiv preprint arXiv:2304.07590* (2023).

Christof Ebert and Panos Louridas. 2023. Generative AI for Software Practitioners. *IEEE Software* 4 (2023), 30–38. https://doi.org/10.1109/MS.2023.3265877

Günes Erkan and Dragomir R Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research* (2004), 457–479. https://doi.org/10.1613/jair.1523

Sen Fang, Tao Zhang, You-Shuai Tan, Zhou Xu, Zhi-Xin Yuan, and Ling-Ze Meng. 2022. PRHAN: automated pull request description generation based on hybrid attention network. *Journal of Systems and Software* (2022), 111160. https://doi.org/10.1016/j.jss.2021.111160

Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, and Michael R Lyu. 2023. Constructing Effective In-Context Demonstration for Code Intelligence Tasks: An Empirical Study. *arXiv preprint arXiv:2304.07575* (2023).

Henry Gilbert, Michael Sandborn, Douglas C Schmidt, Jesse Spencer-Smith, and Jules White. 2023. Semantic Compression With Large Language Models. *arXiv preprint arXiv:2304.12512* (2023).

GitHub Next. 2023. GitHub Next | Copilot for Pull Requests — githubnext.com. https://githubnext.com/projects/copilot-for-pull-requests. [Accessed 23-09-2023].

Mehdi Golzadeh, Alexandre Decan, and Natarajan Chidambaram. 2022. On the accuracy of bot detection techniques. In *Proceedings of the Fourth International Workshop on Bots in Software Engineering*. 1–5. https://doi.org/10.1145/3528228.3528406

Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. [n. d.]. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software* ([n. d.]). https://doi.org/10.1016/j.jss.2021.110911

Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*. 285–296. https://doi.org/10.1145/2884781.2884826

Jian Gu, Pasquale Salza, and Harald C Gall. 2022. Assemble foundation models for automatic code summarization. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 935–946. https://doi.org/10.1109/SANER53432.2022.00112

Jens Hainmueller. 2012. Entropy balancing for causal effects: A multivariate reweighting method to produce balanced samples in observational studies. *Political analysis* 1 (2012), 25–46. https://doi.org/10.1093/pan/mpr025

Hideaki Hata, Nicole Novielli, Sebastian Baltes, Raula Gaikovina Kula, and Christoph Treude. 2022. GitHub Discussions: An Exploratory Study of Early Adoption. *Empirical Software Engineering* 1 (2022), 32 pages. https://doi.org/10.1007/s10664-021-10058-6

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv preprint arXiv:2308.10620* (2023).

Ivana Clairine Irsan, Ting Zhang, Ferdian Thung, David Lo, and Lingxiao Jiang. 2022. AutoPRTitle: A tool for automatic pull request title generation. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 454–458. https://doi.org/10.1109/ICSME55016.2022.00058

Shuyang Jiang, Yuhao Wang, and Yu Wang. 2023. SelfEvolve: A Code Evolution Framework via Large Language Models. *arXiv preprint arXiv:2306.02907* (2023).

Sungmin Kang, Bei Chen, Shin Yoo, and Jian-Guang Lou. 2023. Explainable Automated Debugging via Large Language Model-driven Scientific Debugging. *arXiv preprint arXiv:2304.02195* (2023).

Junaed Younus Khan and Gias Uddin. 2022. Automatic detection and analysis of technical debts in peer-review documentation of r packages. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 765–776. https://doi.org/10.1109/SANER53432.2022.00094

Oleksii Kononenko, Tresa Rose, Olga Baysal, Michael Godfrey, Dennis Theisen, and Bart De Water. 2018. Studying pull request merges: A case study of shopify's active merchant. In *Proceedings of the 40th international conference on software engineering: software engineering in practice*. 124–133. https://doi.org/10.1145/3183519.3183542

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*. 18319–18345.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

Fan Li, Kari Lock Morgan, and Alan M Zaslavsky. 2018. Balancing covariates via propensity score weighting. *J. Amer. Statist. Assoc.* 521 (2018), 390–400. https://doi.org/10.1080/01621459.2016.1260466

Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2023a. Enabling Programming Thinking in Large Language Models Toward Code Generation. *arXiv preprint arXiv:2305.06599* (2023).

Zongjie Li, Chaozheng Wang, Zhibo Liu, Haoxuan Wang, Dong Chen, Shuai Wang, and Cuiyun Gao. 2023b. Cctest: Testing and repairing code completion systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1238–1250. https://doi.org/10.1109/ICSE48619.2023.00110

Zhixing Li, Yue Yu, Tao Wang, Yan Lei, Ying Wang, and Huaimin Wang. 2022. To follow or not to follow: Understanding issue/pull-request templates on github. *IEEE Transactions on Software Engineering* 4 (2022), 2530–2544. https://doi.org/

10.1109/TSE.2022.3224053

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, Xiaohong Zhang, and Meng Yan. 2023a. Improving ChatGPT Prompt for Code Generation. *arXiv preprint arXiv:2305.08360* (2023).

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023b. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210* (2023).

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345* (2019).

Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 176–188. https://doi.org/10.1109/ASE.2019.00026

Antonio Mastropaolo, Luca Pascarella, and Gabriele Bavota. 2022. Using deep learning to generate complete log statements. In *Proceedings of the 44th International Conference on Software Engineering*. 2279–2290. https://doi.org/10.1145/3510003.3511561

Antonio Mastropaolo, Simone Scalabrino, Nathan Cooper, David Nader Palacio, Denys Poshyvanyk, Rocco Oliveto, and Gabriele Bavota. 2021. Studying the usage of text-to-text transfer transformer to support code-related tasks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 336–347. https://doi.org/10.1109/ICSE43902.2021.00041

Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* (2016), 2146–2189. https://doi.org/10.1007/s10664-015-9381-9

Mockus and Votta. 2000. Identifying reasons for software changes using historic databases. In *Proceedings 2000 International Conference on Software Maintenance*. 120–130. https://doi.org/10.1109/ICSM.2000.883028

Nathalia Nascimento, Paulo Alencar, and Donald Cowan. 2023. Comparing Software Developers with ChatGPT: An Empirical Investigation. *arXiv preprint arXiv:2305.11837* (2023).

Marcel Ochs, Krishna Narasimhan, and Mira Mezini. 2023. Evaluating and improving transformers pre-trained on ASTs for Code Completion. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 834–844. https://doi.org/10.1109/SANER56733.2023.00096

Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2023. Examining zero-shot vulnerability repair with large language models. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2339–2356. https://doi.org/10.1109/SP46215.2023.10179324

Rohith Pudari and Neil A Ernst. 2023. From Copilot to Pilot: Towards AI Supported Software Development. *arXiv preprint arXiv:2303.04142* (2023).

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 1 (2020), 5485–5551.

Paul R Rosenbaum and Donald B Rubin. 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika* 1 (1983), 41–55. https://doi.org/10.1093/biomet/70.1.41

Donald B Rubin. 2001. Using propensity scores to help design observational studies: application to the tobacco litigation. *Health Services and Outcomes Research Methodology* (2001), 169–188. https://doi.org/10.1023/A:1020363010465

Oussama Ben Sghaier and Houari Sahraoui. 2023. A Multi-Step Learning Approach to Assist Code Review. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 450–460. https://doi.org/10.1109/SANER56733.2023.00049

Giriprasad Sridhara, Sourav Mazumdar, et al. 2023. ChatGPT: A Study on its Utility for Ubiquitous Software Engineering Tasks. *arXiv preprint arXiv:2305.16837* (2023).

Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2016. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Proceedings of the 38th international conference on software engineering*. 1039–1050. https://doi.org/10.1145/2884781.2884852

Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2017. Review participation in modern code review: An empirical study of the android, Qt, and OpenStack projects. *Empirical Software Engineering* (2017), 768–817. https://doi.org/10.1007/s10664-016-9452-6

Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*. 356–366. https://doi.org/10.1145/2568225.2568315

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* (2017).

Anthony J Viera, Joanne M Garrett, et al. 2005. Understanding interobserver agreement: the kappa statistic. *Fam med* 5 (2005), 360–363.

Dakuo Wang, Elizabeth Churchill, Pattie Maes, Xiangmin Fan, Ben Shneiderman, Yuanchun Shi, and Qianying Wang. 2020. From human-human collaboration to Human-AI collaboration: Designing AI systems that can work together with people. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*. 1–6. https://doi.org/10.1145/3334480.3381069

Dong Wang, Tao Xiao, Patanamon Thongtanunam, Raula Gaikovina Kula, and Kenichi Matsumoto. 2021. Understanding shared links and their intentions to meet information needs in modern code review: A case study of the OpenStack and Qt projects. *Empirical Software Engineering* (2021), 1–32. https://doi.org/10.1007/s10664-021-09997-x

Jian Wang, Shangqing Liu, Xiaofei Xie, and Yi Li. 2023. Evaluating AIGC Detectors on Code Content. *arXiv preprint arXiv:2304.05193* (2023).

Yi Wu, Nan Jiang, Hung Viet Pham, Thibaud Lutellier, Jordan Davis, Lin Tan, Petr Babkin, and Sameena Shah. 2023. How Effective Are Neural Networks for Fixing Security Vulnerabilities. *arXiv preprint arXiv:2305.18607* (2023).

Zhuohao Wu, Danwen Ji, Kaiwen Yu, Xianxu Zeng, Dingming Wu, and Mohammad Shidujaman. 2021. AI creativity and the human-AI co-creation model. In *Human-Computer Interaction. Theory, Methods and Tools: Thematic Area, HCI 2021, Held as Part of the 23rd HCI International Conference, HCII 2021, Virtual Event, July 24–29, 2021, Proceedings, Part I 23*. 171–190. https://doi.org/10.1007/978-3-030-78462-1_13

Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023)*. https://doi.org/10.1109/ICSE48619.2023.00129

Chunqiu Steven Xia and Lingming Zhang. 2023. Conversational automated program repair. *arXiv preprint arXiv:2301.13246* (2023).

Tao Xiao, Hideaki Hata, Christoph Treude, and Kenichi Matsumoto. 2024. *Research Artifact - Generative AI for Pull Request Descriptions: Adoption, Impact, and Developer Interventions*. https://doi.org/10.5281/zenodo.10656106

Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10. https://doi.org/10.1145/3520312.3534862

Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. *arXiv preprint arXiv:2304.10778* (2023).

Zhengran Zeng, Hanzhuo Tan, Haotian Zhang, Jing Li, Yuqun Zhang, and Lingming Zhang. 2022. An extensive study on pre-trained models for program understanding and generation. In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis*. 39–51. https://doi.org/10.1145/3533767.3534390

Xunhui Zhang, Yue Yu, Georgios Gousios, and Ayushi Rastogi. 2022. Pull request decisions explained: An empirical overview. *IEEE Transactions on Software Engineering* 2 (2022), 849–871. https://doi.org/10.1109/TSE.2022.3165056

Received 2023-09-29; accepted 2024-01-23