GitHub Copilot: a systematic study

Alessandro Benetti[1,*], Michele Filannino[1]

[1]Prometeia, Piazza Trento e Trieste, 30 - Bologna, 40137, Italy

Abstract

This paper examines the effects of GitHub Copilot, a prominent example of generative artificial intelligence (GAI), on software development methodologies. Through an empirical study of GitHub Copilot's performance in a professional setting, we assess its value across various programming environments. Our comprehensive evaluation reveals that GitHub Copilot significantly improves developer productivity and assistance in different coding scenarios. Furthermore, the research outlines effective strategies for leveraging GitHub Copilot to its fullest potential, thus advancing the use of GAI tools in software engineering. While recognizing GitHub Copilot's considerable advantages, we also identify its shortcomings and areas in need of further improvement.

Keywords

Generative AI, Software Engineering, GitHub, Software Development, Systematic Study, GAI, Coding Assistance

1. Introduction

The advent of Generative Artificial Intelligence (GAI) is transforming our approach to creativity and the production of new content. GAI encompasses machine learning algorithms capable of generating content—ranging from images, videos, and text to music—that mirrors the style and quality of human-created works.

Recent breakthroughs in deep learning have given rise to sophisticated GAI models, such as latent diffusion models [1] and Generative Pre-trained Transformers (GPT) [2]. These models, capable of producing realistic and varied content with minimal human oversight, are trained on extensive datasets and generate new items by sampling from a learned probability distribution.

GAI's potential is vast, with applications including the creation of lifelike virtual imagery (e.g., DALLE-3 [3], Midjourney [4], Stable Diffusion [5]), serving as efficient writing assistants or conversational agents (e.g., ChatGPT [6], LLAMA [7], Gemini [8], Claude [9]). However, the rapid adoption of GAI technologies necessitates careful consideration of their ethical and responsible use, particularly in light of significant ethical and legal challenges such as intellectual property rights, privacy issues, and the potential for misuse of GAI-generated content.

2. GitHub Copilot

GitHub Copilot distinguishes itself as an innovative application of GAI, offering substantial assistance to developers in coding tasks. It is based on a GPT-3.5 model, which is part of the advanced Generative Pre-trained Transformer series. This model leverages the Transformer architecture [10], renowned for its efficacy in processing and generating text that closely resembles human writing. GPT-3.5's capabilities extend to a comprehensive understanding of language subtleties, contextual nuances, and notably, programming code syntax.

GitHub Copilot's primary objective is to enable programmers to concentrate on problem-solving instead of searching for the appropriate libraries and functions to implement their desired solution. With this tool, programmers can unlock a new level of productivity and efficiency and deliver high-quality code in a fraction of the time that it typically takes.

The primary capabilities of GitHub Copilot can be summarized as follows:

- Natural language interface: Developers can interact with GitHub Copilot using natural language commands. This means they can describe what they want to achieve in plain English, and GitHub Copilot will suggest code to accomplish the task.
- Integrated with IDEs: GitHub Copilot is integrated with popular code editors and IDEs, including Visual Studio Code, Visual Studio MSDN, and PyCharm.
- Context-aware: GitHub Copilot analyzes the context of the code being written and generates suggestions accordingly.
- Privacy-focused: GitHub Copilot for Business does not retain telemetry or code snippets data.

While GitHub Copilot can be a powerful tool for developers, it is important to underline some of the potential concerns that are also somewhat common to most Large Language Models:

- Potentially inaccurate code: one potential concern with GitHub Copilot is that it may generate

Ital-IA 2024: 4th National Conference on Artificial Intelligence, organized by CINI, May 29-30, 2024, Naples, Italy

*Corresponding author.
alessandro.benetti@prometeia.com (A. Benetti); michele.filannino@prometeia.com (M. Filannino)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-3762/489.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

incorrect or non-functional code.

- Limited world or codebase knowledge after the training date: this might cause the suggestion of deprecated methods for libraries that change significantly over time.
- No match with the information that the programmer has: this is true for both the overall context of the code that it is suggesting, and some intrinsic knowledge about the world that the programmer has, like awareness among other things.

To address these concerns, it is important for developers to carefully review and test the code generated by GitHub Copilot.

3. Methodology

With the advent of this groundbreaking technology, it is crucial to thoroughly evaluate its potential through extensive testing. At Prometeia, a software development-focused consulting firm, we've decided to embark on a pilot study aimed explicitly at evaluating the functionalities of GitHub Copilot.

We chose GitHub Copilot Business over alternatives like Tabnine, Blackbox, and Sourcery due to its wide range of supported programming languages, compatibility with various Integrated Development Environments (IDEs), and advanced features that meet enterprise standards, including scalability, security, and compliance.

Various reviews and studies, including those by Vaithilingam [11], the GitHub Copilot study [12]. However, these investigations have occasionally encountered contradictory findings and have not specifically concentrated on the implementation of this tool within a real-world corporate environment.

Undertaking a pilot study offers numerous advantages, making it a strategic approach for our evaluation process. Firstly, the pilot allows our developers to assess the tool's effectiveness by testing it on a small scale. This provides an opportunity to gauge how well GitHub Copilot can assist in achieving their objectives, determine if the generated code meets their requirements, and assess if it improves their current development process. Secondly, a pilot study can assist us in identifying any limitations or potential issues with GitHub Copilot, such as difficulties with specific programming languages or complex coding tasks. By identifying such limitations early on, we can avoid potential problems and find alternatives or workarounds to using the tool, thus saving time and money in the long run. Hence, this initiative aimed to determine whether GitHub Copilot would be a viable addition to our software development toolkit.

To compare our findings with other studies, we included most of the key performance indicators (KPIs) tracked in the previous ones. Therefore, we decided to adopt the SPACE framework (Forsgren, 2021 [13]), which focuses on various aspects of developer productivity, ranging from overall individual satisfaction to knowledge sharing among different individuals. A summary of these questions can be found at the following url.

3.1. Participants Selection

For this study, we selected 31 participants from three specialized branches within our company, in particular:

- **Branch A**, a development team of a longstanding software solution, working on both new features and the maintenance of pre-existing ones.
- **Branch B**, focused mostly on the development of a new software product.
- **Branch C**, the development team of a software cloud product, engaged with both development of new features and maintenance.

These participants were selected due to their involvement in a broad range of projects, encompassing both innovative and established (legacy) projects. To promote an unbiased evaluation, we refrained from assigning predetermined tasks, allowing participants to incorporate the tool into their regular workflow. Over a two-month observation period, we monitored their usage of GitHub Copilot, aiming to capture its utility across diverse project types and user experiences. Notably, all of the selected participants had no less than 1 years of programming experience.

In order to gather participant feedback, we organized a series of in-person meetings, offering a forum for them to share their experiences with GitHub Copilot. Based on the insights gained during these discussions, we crafted a 16-question survey covering the SPACE framework dimensions (available at the following url). This survey mixed questions from existing research with new ones specifically designed for this study, including both closed and open-ended questions. The closed-ended questions aimed to collect quantitative data, while the open-ended ones sought to capture more nuanced feedback on their experiences. This approach aimed to collect quantitative and qualitative data to comprehensively evaluate GitHub Copilot's performance.

4. Results & Discussion

The following section presents some of the key findings obtained from the analysis of participant responses. This section will be divided into three parts:

- **overall ratings**: Participants were asked to rate GitHub Copilot on a scale from 1 to 10, with 10 being the highest score. This rating serves as an overall assessment of GitHub Copilot's performance and effectiveness.
- **main benefits**: This section highlights the areas where GitHub Copilot excels. It examines the specific aspects or functionalities of the tool that participants found most valuable or beneficial in their programming tasks.
- **main drawbacks**: In this part we explore the challenges or limitations experienced by participants when using GitHub Copilot. It focuses on the areas where the tool may struggle or encounter difficulties, as reported by the participants.

In the following sections, we will delve deeper into these areas to comprehensively analyse the findings.

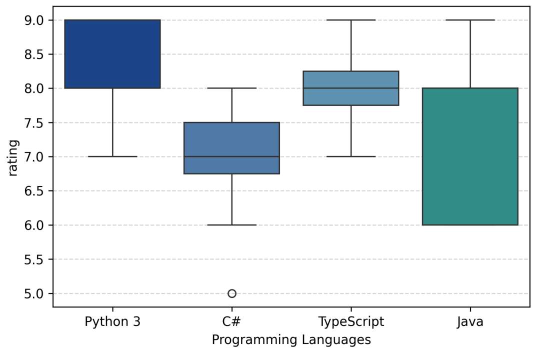

Figure 1: Overall rating given to GitHub Copilot, by programming language

4.1. Overall feedback

The evaluation of GitHub Copilot's efficacy, as illustrated in Figure 1, reveals an overall positive reception, with a computed mean rating of 7.4 on a scale where higher values denote greater approval. This overarching assessment, however, masks underlying variations in user satisfaction that are closely linked to the specific programming language in use by the developers.

A more granular analysis of the data shows that developers employing high-level programming languages, notably Python and Java, tend to assign higher ratings to GitHub Copilot. This distinction suggests a potential correlation between the nature of the programming language and the tool's performance. High-level languages, characterized by their abstraction from machine languages and emphasis on readability, may inherently facilitate more accurate and contextually relevant code suggestions by AI-based tools like GitHub Copilot. Additionally, the extensive availability of open-source code in these languages may provide a richer dataset for the AI's learning algorithms, enhancing its predictive accuracy and relevance. Conversely, the analysis reveals a modest decline in satisfaction among C# developers, with a mean score of 7. This discrepancy hints at possible limitations in GitHub Copilot's adaptability or efficiency across different programming environments. The factors contributing to this variation could range from the structural and syntactical idiosyncrasies of C# that challenge the AI's prediction models, to a potentially lesser volume of training data derived from C# codebases.

These insights advocate for a more nuanced approach to the continuous development and refinement of GitHub Copilot, emphasizing the need for language-specific optimizations to cater to the diverse requirements of the development community. For users, the findings highlight the importance of aligning expectations with the capabilities and limitations of AI tools within specific programming contexts.

4.2. Main benefits

The study's findings, as visualized in Figure 2, delineate the multifaceted benefits that GitHub Copilot offers to developers, underscoring its impact on productivity and code quality.

One of the principal advantages identified by participants is Copilot's proficiency in auto-generating **boilerplate code** and foundational code structures. This feature significantly reduces the time and effort required during the initial phases of project setup, allowing developers to bypass the tedium of crafting repetitive code patterns from scratch. Such efficiency in establishing project infrastructure is not only a time-saver but also enables a smoother transition to more complex development tasks.

Moreover, GitHub Copilot's contribution to **code documentation** represents another vital benefit. The tool's ability to furnish quick and precise descriptions for functions, classes, and various code segments assists developers in maintaining well-documented codebases. Proper documentation is crucial for enhancing code readability, facilitating easier maintenance, and enabling smoother collaboration among team members. By automating this aspect, Copilot aids in ensuring that projects adhere to best practices in code documentation, thus elevating the overall quality of the development process.

The generation of **test code** for existing functions by GitHub Copilot is highlighted as a particularly advantageous feature. This capability assists developers in creating comprehensive test suites, a critical component of the software development life cycle aimed at verifying the correctness and reliability of code. Notably, we ad-

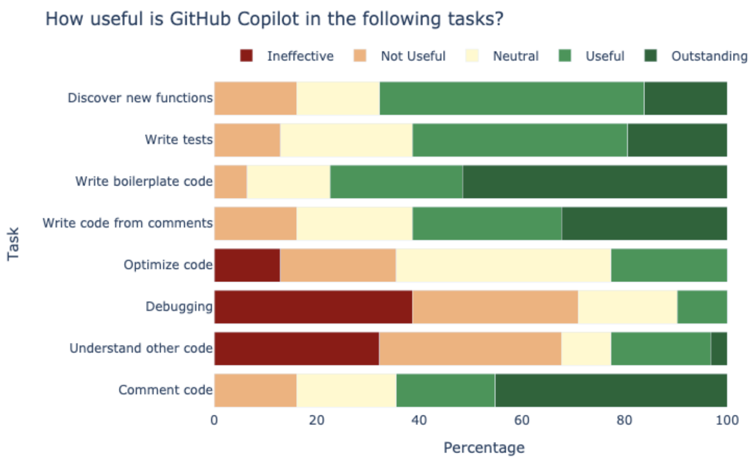

Figure 2: GitHub Copilot usefulness on different tasks

vised the developers to exercise increased caution when incorporating tests authored by GitHub Copilot, given their significant influence on the code's overall reliability.

An interesting observation from the study is the strategic utilization of time saved through GitHub Copilot's assistance. Many participants reported reallocating the time gained to enhance the quality of their products further by focusing on rigorous testing, refining documentation, or dedicating effort to areas of the project that could benefit from manual oversight. Alternatively, some participants chose to invest the saved time into personal development, such as exploring new programming libraries, learning new tools, or contributing to other projects. This flexibility underscores Copilot's role not just as a tool for immediate productivity gains but also as an enabler for broader professional growth and product quality enhancement.

4.3. Main drawbacks

While GitHub Copilot has been lauded for its ability to enhance developer productivity and streamline workflows, the tool is not without its limitations, which can impact its overall effectiveness in certain contexts.

One notable concern is its integration with Integrated Development Environments (IDEs), particularly when used alongside other coding aids such as Intellisense. Some users have reported **conflicts between GitHub Copilot and Intellisense**, leading to potential confusion and errors. This issue underscores the importance of seamless tool integration within the development environment to prevent disruption in the coding process.

Another drawback observed by users is the tool's tendency to **suggest repetitive code**. Such suggestions can potentially lead to less efficient or elegant coding solutions, contradicting the tool's aim to streamline development efforts. This behavior might stem from the AI's training data or its current understanding of best coding practices, indicating an area for further refinement to ensure that Copilot consistently proposes high-quality and contextually appropriate code.

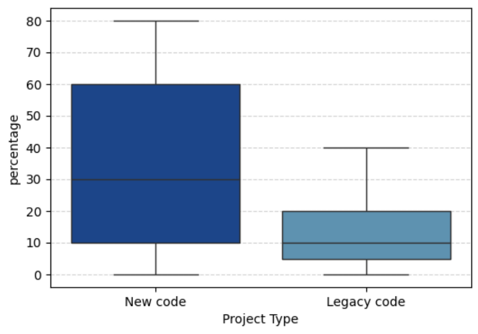

Figure 3: Percentage of code lines written by GitHub Copilot

The effectiveness of GitHub Copilot appears to vary significantly when dealing with different types of codebases. Specifically, its performance with large or legacy codebases presents challenges, as evidenced by a reported median contribution of merely 10% (Figure 3) to the lines of code written in such contexts. This reduced effectiveness could be attributed to the AI's limited ability to fully comprehend the complexities and nuances of older or more extensive codebases, leading to challenges in generating accurate and useful code suggestions.

Conversely, GitHub Copilot demonstrates considerably greater efficiency with new codebases, where it contributes to around 30% of the written code. This discrepancy highlights Copilot's aptitude for aiding in the rapid development of new projects, where its capabilities in generating boilerplate code and structuring new projects can be most beneficial.

5. Related works

The assessment of Artificial Intelligence (AI) tools' impact on various sectors, particularly in software development, has been an area of interest in the recent years. The advent of AI innovations has been consistently associated with enhancements in productivity levels and the facilitation of a more intuitive process for coding, as documented in the findings of Chen et al. (2021) [14], who illustrated the positive ramifications of AI on software engineering practices.

In the realm of code generation, the deployment of deep learning methodologies, especially those utilizing Transformer models, has been met with considerable success. A noteworthy illustration of this is the study by Feng et al. (2020) [15], which presents a model that significantly surpasses the efficiency of conventional code completion tools. This approach, which harnesses the power of deep learning to achieve a contextual comprehension and prediction, showcases the potential of generative AI to navigate and replicate complex coding idioms and patterns with remarkable accuracy.

Furthermore, research conducted by Yetistiren et al. (2022) [16] denotes the proficiency of GitHub Copilot in understanding coding syntax. This underlines the broad spectrum of advantages offered by AI in the realm of software development, extending beyond mere procedural improvements to include significant qualitative enhancements in code management and optimization.

Despite the proliferation of studies and reviews in this area, it is crucial to acknowledge the predominance of *in vitro* research methodologies (where specific programming tasks are assigned to participants) and the occasional emergence of conflicting findings, as highlighted by Vaithilingam (2022) [11] and the study on GitHub Copilot (2021) [12]. These discrepancies underscore the necessity for our own comprehensive *in vivo* study (in which no specific programming tasks are prescribed), involving a diverse array of developers from various corporate sectors, employing different Integrated Development Environments (IDEs), and programming languages.

5.1. Comparison with other studies

To further understand the impact of GitHub Copilot, we incorporated a key performance indicator (KPI) from a GitHub Copilot survey [12] for a direct comparison with our study's outcomes. Our findings, reported in Table 1 showed notable differences from GitHub's reported results. Although our results still reflect a highly positive sentiment, it is important to note that the differences may be attributed to the nature of the experiments conducted by GitHub. Specifically, our study population consisted of individuals with impending deadlines, which could influence their perceptions and experiences with the tool.

	GitHub	Prometeia			
Question	Overall	Overall	Branch A	Branch B	Branch C
Focus on more satisfying work	74%	35%	36%	44%	27%
Feel more productive	88%	32%	18%	44%	36%
Are faster with repetitive tasks	96%	74%	82%	78%	63%

Table 1
GitHub study performance metrics

The variations in percentages observed between the groups can be ascribed to various factors, such as the programming languages employed by developers and the nature of the projects they were engaged in. For example, diverse branches may adopt distinct programming practices, preferences, and project requirements, which can shape their views and usage of GitHub Copilot. Additionally, the nature of the projects (whether new or legacy) can also impact how the benefits and drawbacks of GitHub Copilot are perceived. These factors highlight the significance of considering the context in which the tool is employed and comprehending its potential influence on the recorded percentages.

6. Conclusions

The study revealed a generally positive overall rating for GitHub Copilot, with developers giving it an average score of 7.4 (out of 10) and, as expected, ratings varied based on the programming language used. Our developers identified several benefits of using GitHub Copilot. One of its main advantages is the ability to generate boilerplate and basic code structures, saving developers time and effort during project setup. Additionally, the tool ensures proper code documentation by providing accurate descriptions of functions, classes, and other code elements. Another notable benefit is its capability to generate test code for existing functions, contributing to code reliability. Coherently with its original goal, this study proved GitHub Copilot effective in supporting our software development activities. As a result, various branches of our company started using the tool as part of their development standard toolkit. Lastly, we could not evaluate the tool on junior programmers, which leaves an area of inquiry for future studies. Understanding how newcomers to the field, with potentially different learning curves and development practices, interact with and benefit from GitHub Copilot could provide valuable insights into its overall utility and areas for improvement.

Acknowledgments

The authors wish to express their sincere gratitude to Dave Burnison from GitHub for his feedback on the Copilot service. His insights and expertise have significantly contributed to the research and understanding of the impact of AI-assisted programming tools in software development.

References

[1] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution image synthesis with latent diffusion models, 2022. arXiv:2112.10752.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, 2020. arXiv:2005.14165.

[3] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo, et al., Improving image generation with better captions, Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf 2 (2023) 8.

[4] Midjourney, https://www.midjourney.com/home, 2024. Accessed: March 25, 2024.

[5] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, et al., Scaling rectified flow transformers for high-resolution image synthesis, arXiv preprint arXiv:2403.03206 (2024).

[6] Chatgpt, https://openai.com/blog/chatgpt, 2024. Accessed: April 4, 2024.

[7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, Llama: Open and efficient foundation language models, 2023. arXiv:2302.13971.

[8] Gemini-Team, Gemini: A family of highly capable multimodal models, 2024. arXiv:2312.11805.

[9] Claude, https://www.anthropic.com/claude, 2024. Accessed: April 4, 2024.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2023. arXiv:1706.03762.

[11] P. Vaithilingam, T. Zhang, E. L. Glassman, Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, CHI EA '22, Association for Computing Machinery, New York, NY, USA, 2022. URL: https://doi.org/10.1145/3491101.3519665. doi:10.1145/3491101.3519665.

[12] Research: Quantifying GitHub Copilot's impact on developer productivity and happiness, 2022. Accessed: 2024-03-13. GitHub Copilot Study.

[13] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, J. Butler, The SPACE of developer productivity: There's more to it than you think., Queue 19 (2021) 20–48. doi:10.1145/3454122.3454124.

[14] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, W. Zaremba, Evaluating large language models trained on code, 2021. arXiv:2107.03374.

[15] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, Codebert: A pre-trained model for programming and natural languages, 2020. arXiv:2002.08155.

[16] B. Yetistiren, I. Ozsoy, E. Tuzun, Assessing the quality of GitHub Copilot's code generation, in: Proceedings of the 18th international conference on predictive models and data analytics in software engineering, 2022, pp. 62–71.