Taking Flight with Copilot

CHRISTIAN BIRD,
DENAE FORD,
THOMAS ZIMMERMANN,
NICOLE FORSGREN,
EIRINI KALLIAMVAKOU,
TRAVIS LOWDERMILK,
IDAN GAZIT

EARLY INSIGHTS AND OPPORTUNITIES OF AI-POWERED PAIR-PROGRAMMING TOOLS

In pair programming, two developers write code together. One takes the role of *driver*, writing the code needed for the task at hand, while the other assumes the role of *navigator*, directing the driver's work and reviewing the code. This allows the driver to focus on detailed coding—syntax and structure—while letting the navigator direct the flow of the work and review the code in realtime. Proponents of pair programming say it improves code quality and readability, and can speed up the reviewing process.[5] Effective pair programming to date, however, has required the complex coordination of getting two programmers to work together synchronously. This has made it challenging for teams to adopt this approach at scale. The emergence of new AI-powered tools to support programmers has shifted what it means to pair program.

GitHub's Copilot is an AI-powered developer tool leading this shift. GitHub released Copilot in a complimentary technical preview on June 29, 2021, letting hundreds of thousands of developers try coding with an AI pair programmer. (Copilot became generally available as a paid product on June 21, 2022. This article focuses on the earliest

releases of Copilot (the free technical preview), because these allowed us to capture some of the first experiences with an AI pair programmer. While some changes to Copilot have been made in the version released in 2022, the user interface and experience is largely the same.)

With developers taking the role of navigator, they can direct the detailed development work and review the code as it is written. In addition, the AI assistant can write code (directed by the developer navigator) much faster than a peer, potentially speeding up the process.

Copilot received public attention quickly, generating conversation in forums, press, and social media. These impressions ranged from excitement about potentially boosting developers' productivity to apprehension about AI replacing programmers in their jobs.

But what were developers' actual experiences with Copilot, beyond the hype found in top tweets, Hacker News, or Reddit posts? During the early days of the technical preview, we investigated the initial experiences of Copilot users, presenting the unique opportunity to watch how developers would use it, as well as what challenges they encountered. While most of the observations were expected, there were some surprises as well. This article presents highlights from these initial empirical investigations. (The models and technology used to develop Copilot are changing rapidly. This analysis was current as of January 2022. Although some features of the tool have evolved, the general themes discussed here still hold true at the date of publication.)

These highlights are:

- The diverse ways that developers are using Copilot,

including things that worked great and things that very much didn't.
- Challenges that developers encountered when using the technical preview of Copilot, yielding insights into how to improve AI pair-programming experiences.
- How AI pair programming shifts the coding experience and reveals the importance of different skills—for example, the emerging importance for developers to know how to *review* code as much as to *write* code.
- A discussion of opportunities that AI presents to the software development process and its potential impact.

We conducted three studies to understand how developers use Copilot:
- An analysis of forum discussions from early Copilot users.
- A case study of Python developers using Copilot for the first time.
- A large-scale survey of Copilot users to understand its impact on productivity.

Each of these studies (summarized in table 1) is discussed here, following an introduction to Copilot.

WHAT IS COPILOT AND HOW DOES IT WORK

At its core, Copilot consists of a large language model and an IDE (integrated development environment) extension that queries the model for code suggestions and displays them to the user. You may already have interacted with such models when writing documents or text messages. These models have been trained on billions of lines of text and have developed the ability to predict, with high

TABLE 1: **SUMMARY OF STUDIES INCLUDED IN THIS PAPER**

STUDY 1: FORUM ANALYSIS Oct/Nov 2021	**STUDY 2: CASE STUDY** Dec/Jan 2022	**STUDY 3: SURVEY** Feb/Mar 2022
What are people using Copilot for?	How are first-timers engaging with Copilot?	How does Copilot impact productivity?
Approach: Reviewed and analyzed 279 GitHub Discussions forum posts	Approach: Conducted a think-aloud study with 5 expert Python developers	Approach: Analysis of 2047 survey responses from Copilot developers
Key Finding: Participants reported spending less time on Stack Overflow, but now have less of an understanding of how or why the code works.	Key Finding: Participants accept the suggestion for efficiency but give up a small bit of autonomy/control over the code they're writing. We observed participants wrestle with this in real-time.	Key Finding: We were able to correlate 11 usage metrics to perceived productivity. Acceptance rate had the highest positive correlation to aggregate perceived productivity.

accuracy, what you are going to type next.

The important difference here is that Copilot uses Codex, a language model trained on source code instead of text (e.g., emails, text messages, websites, or documents). This source code came from a large portion of the public code on GitHub, the largest repository of source code in existence.

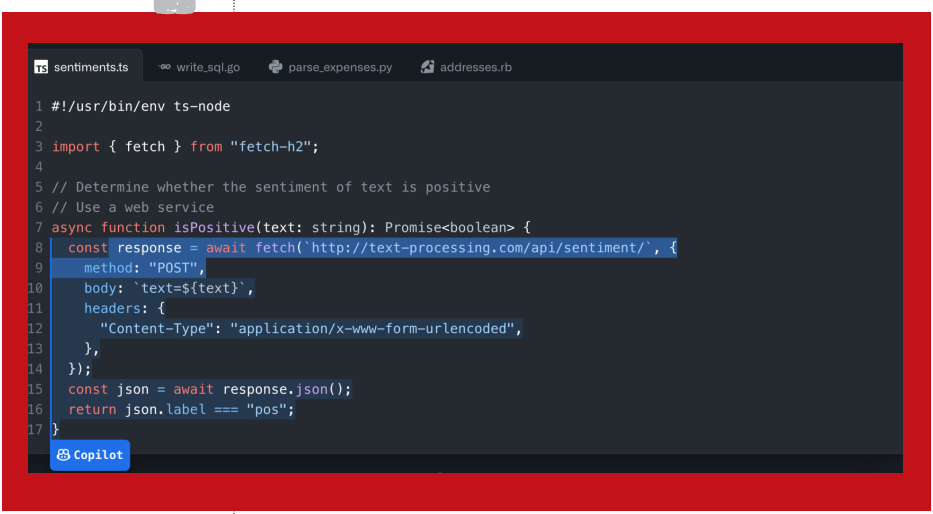
According to OpenAI, the team that built Codex, the model has been trained on more than 159 GB of Python code alone, in addition to code from many other languages. As such, it's quite common for code that a developer is writing to be similar to some combination of pieces of code that Copilot has seen before (during model training). Copilot recognizes these similarities and offers suggestions based on the similar pieces of code on which it was trained. Copilot, however, doesn't make suggestions to developers by making copies of code that it has seen previously. Rather, Copilot

generates new suggestions by synthesizing what it has observed in billions of lines of code during model training (and that may not even exist in any code base).

While Copilot leverages this very large model, more than a high-quality code-suggestion engine is required to help developers be more productive. Copilot has been incorporated into multiple IDEs in a way that makes code suggestions timely and seamless. As you write code, requests are continuously sent to Copilot's AI model, which is optimized to analyze the code, identify useful suggestions, and send them back in fractions of a second so that they can be offered to developers in their IDEs when needed, as shown in figure 1.

The experience developers encounter is similar to the existing autocompletion that has been in modern IDEs

FIGURE 1: **EXAMPLE SCREENSHOT OF IDE WITH COPILOT CODE COMPLETION**



for years. The difference is that these suggestions may be longer, sometimes spanning multiple lines of code, and ideally more contextually relevant and helpful. Copilot can currently be used in VS Code, Visual Studio, Neovim, and JetBrains IDEs, including PyCharm and IDEA.

During these initial investigations of Copilot, developers showed they were unclear about how it worked. Among the most common misunderstandings was that while Copilot does learn from code, the learning happens during a general training phase, where OpenAI trains a general-purpose programming model (Codex) that is fine-tuned by using a selected set of public codebases on GitHub. (For more information about Codex, a model fine-tuned by OpenAI, refer to https://openai.com/blog/openai-codex/.)

By contrast, when a developer uses Copilot to generate suggestions during a coding session, the code sent to Copilot to generate the suggestion is not used to make suggestions to other developers. Simply put, this model will not "leak" code to anyone else. In fact, such web requests are ephemeral, and the code is not logged, captured, or recorded (unless users agree to collection).

WHAT ARE PEOPLE USING COPILOT FOR?

To get a better understanding of how people were using Copilot, we collected and analyzed users' self-reported experiences to uncover the challenges they faced and the opportunities this tool presented them, as well as to identify how the technology could evolve. The Copilot Discussion forum, available to all technical preview users, was a valuable resource. In this forum, Copilot users provided code snippets, links to blogs, and even live coding

videos. An analysis of all 279 posts available, which were devoted to general feedback and personal showcases, uncovered strengths and challenges of Copilot, as well as uses that extend beyond traditional coding tasks.

Among the strengths of Copilot highlighted by users: It helped them adapt to different writing styles, generate simple methods and classes, and even infer which methods to write based on comments. This flexibility is largely a result of the context used by Copilot to make its recommendations. It uses information about the currently active file and code region to prompt the Codex model and adapt the recommendations to developers. Users reported that Copilot can adapt to different coding styles (i.e., naming, formatting). In one instance, Copilot was able to make recommendations in someone's own custom programming language. The ability to work with multiple languages was also cited as a strength.

Challenges that Copilot users raised included the risk of revealing secrets such as API keys and passwords or suggesting inappropriate text. This feedback was addressed during the technical preview by adding a filter to Copilot that removed problematic suggestions. Users reported that Copilot sometimes does not write "defensive code"—for example, when Copilot writes a complete method that takes a parameter, it may not check if pointers are null or if array indices are negative. In some cases, users found Copilot suggestions to be distracting and, as such, requested keyboard shortcuts for turning Copilot on/off or muting it for a period of time. Users also asked that the context should take more than the current file into account.

In addition to traditional AI pair programming (that is,

code completion that people think of when using AI in a programming environment), Copilot was used for tasks that extended well beyond this scenario. A few interesting use cases emerged where participants used Copilot to help them learn a new programming language, set up a quick server, calculate math problems, and assist in writing unit tests and regular expressions.

These first examples are related to coding but are much more powerful than simple line completion. In fact, one developer passed two coding skills tests without prior knowledge of complex code in the tested languages by using knowledge of other languages and leveraging Copilot for support. Several developers used Copilot to translate text messages from one language to another—for example, from English to French. Another developer connected Copilot with a speech-recognition tool to improve accessibility by building a code-as-you-speak feature. The range of user experiences we observed during the technical preview of Copilot are summarized with examples in table 2.

Some early users had questions about how Copilot creates its code suggestions. They expressed uncertainty about licensing the code generated by Copilot. Some were of the opinion that aspects of Copilot (such as code suggestions) might be required to be released under open licenses, because the code used to train Codex (the AI model powering Copilot's suggestions) was potentially subject to open-source license. Others chimed in with different perspectives, overall expressing different opinions on whether Copilot's suggestions should be released under open licenses (or if used by developers,

TABLE 2: **EARLY EXPERIENCES WITH COPILOT**

THE GOOD	THE BAD	THE MIXED
users reported productivity improvements	users report Copilot gets into loops of suggesting the same thing	less coding but more reviewing
users compared Copilot to a human	Copilot doesn't write "defensive code" e.g., check null pointers	less time on Stack Overflow, but now less understanding of how/why the code works
"Copilot removes the ceiling on creativity"	Copilot sometimes suggests inappropriate text	API discoverability is supported but doesn't provide enough information to select best solution
some said Copilot suggested something they would "ordinarily overlook"	Copilot at times leaks PII in header files	

could pose license compliance issues based on inclusion of code suggestions). The rationale for these opinions varied: Some believed that open-source licenses applicable to code used to train Copilot somehow applied; others pointed to copyright law; and still others posited a "moral basis" for giving back to open-source developers.

There were also discussions about how copyright applied to Copilot's code suggestions. Some users questioned whether code suggested by Copilot would be protected by copyright, and if so, who might assert the copyright—GitHub, a developer using Copilot on a project, or even owners of code used to train Copilot (in cases where Copilot's suggestions matched the code used to train Copilot). The discussions revealed that many developers may be unfamiliar with global copyright laws. Similarly, the discussions revealed that developers have differing

perspectives about whether code generated by use of an AI model should be capable of copyright protection.

(Readers should note that this section summarizes the comments in Copilot Discussion forum posts and that users did not use precise terms. Also note that this summary presents a synopsis of the posts as they were observed and does not represent any personal opinions of the authors or official stance of the authors' employers. Finally, note that while we have endeavored to summarize the comments to our best ability, we are not legal professionals and this summary is not intended to serve as a legal review of these topics.)

Study Protocol

The Copilot case study took place over two days. The team spent an hour with each participant. Each interview took place over Microsoft Teams. The researchers used a PowerPoint presentation to organize the study, and participants were asked to share their screens as they engaged with Copilot.

Each interview consisted of the following steps:

Task 1: Small Demonstration of Copilot

Participants were asked to launch Copilot via a preconfigured Codespace linked to a GitHub repository. From there participants were asked to complete a basic function that accepts an integer and returns whether that integer is a prime number. As discoverability was not part of the study, descriptions and guidance were provided, pointing out features of Copilot as they presented themselves.

HOW ARE FIRST-TIME USERS ENGAGING WITH COPILOT?

Findings from the forum analysis inspired several questions about Copilot use in practice. Thus, to better understand how developers engage with this tool in situ, we conducted an in-depth case study with five professional Python developers, strategically selected as external industry developers who had not

Task 2: Creating a Command-line Tic-Tac-Toe Game

Once participants understood how to engage Copilot and its basic functionality, they were asked to build a tic-tac-toe game with the following criteria:

1. Add a class for a tic-tac-toe board that defines the nine cells of the game board.
2. Add a method that accepts and tracks player moves on the board.
3. Add a method that displays the board in the command line.
4. Add a method that determines when the game is over and whether a player has won.
5. Write code to test that the board is displayed correctly and that the game is over when a player wins.

Task 3: Implement a "Send Email" Feature

To get a sense of how participants would respond to using an unfamiliar API (and how Copliot would respond with suggestions), they were asked to build a feature to email the researcher when a game is complete. All participants noted that they had never used the smtplib library before.

Post-Study Reflective Interview

After roughly 35-40 minutes of using Copilot to complete these tasks, participants were asked a set of rating-scale questions, interacted with Copilot before. The study protocol was as follows:

1. Participants were walked through a short demonstration of Copilot.
2. Participants were asked to implement a command-line tic-tac-toe game following a short description of the requirements.
3. Participants were asked to implement a "Send email" feature."
4. A post-study reflective interview was conducted to assess the participants' overall perspective on Copilot.

The order of these tasks was chosen strategically: First to provide guidance to the participants on how to use Copilot; then have them use the tool to independently build their own foundation and mental models of how things work; and finally to create a scenario where they would likely need to look up an API they do not

concluding with one open-ended question:

1. How would you rate your overall experience with Copilot today? (1 – "Not at all positive" to 5 – "Very positive")
2. How helpful was Copilot while you attempted to complete the tasks today? (1 – "Not at all helpful" to 5 – "Very helpful")
3. How effective were Copilot's suggestions while attempting to complete the tasks today? (1 – "Not at all effective" to 5 – "Very effective")
4. How different were Copilot's suggestions from your typical "style" of writing code? (1 – "Not at all different" to 5 – "Very different")*
5. How interruptive was Copilot while you were trying to complete the tasks today? (1 – "Very interruptive" to 5 – "Not at all interruptive")
6. How likely would you be to recommend Copilot to a friend or colleague? (1 – "Not at all likely" to 5 – "Very likely")
7. How likely would you be to use Copilot if it was available today? (1 – "Not at all likely" to 5 – "Very likely")
8. How disappointed would you be if GitHub decided to discontinue Copilot? (1 – "Not at all disappointed" to 5 – "Very disappointed")
9. How would you describe Copilot to one of your colleagues who has not heard of it before?

use often. We hypothesized that recalling the correct use of APIs is where Copilot can provide additional value because it combines code completion (that is, code that developers would likely be able to provide themselves), as well as additional information that developers might typically need to search for (for example, by searching Stack Overflow). The appendix at the end of this article provides additional details on the study protocol.

At the completion of the case study, researchers convened to discuss participant responses and common themes that emerged. Participants' experiences ranged from enjoyment, guilt, skepticism, clarity on how to interact with the tool, and opportunities for AI pair programming to evolve.

In terms of enjoyment, many participants expressed overall amazement during

their first interactions with the tool: "*Wow. That saved me a heck of a lot of time. Yeah, I think I would've tried to do it all in one line and throw some line breaks in there. But this looks better and it's actually a bit more readable.*"

Following this first interaction, the light guilt of having this tool at their disposal set in for some participants. One described it as: "*This is pretty cool. At the same time, [Copilot can] make you a little lazy when thinking about the logic you want to implement. You'll just focus on that logic, which it's providing to you, instead of thinking of your own [logic].*"

A core highlight from this case study was the early indicator that developers using AI-assisted tools often spend more time reading and reviewing code than writing, findings supported by other investigations.[4] The insights from study participants also highlighted some promising areas to investigate in the future, such as including more context around suggestions presented (e.g., which suggestion optimizes for readability, performance, or conciseness), as well as code provenance.

Finally, all of the participants remarked that they felt Copilot contributed to their productivity. This led to further investigation and the final study covered in this article.

HOW DOES COPILOT IMPACT PRODUCTIVITY?

To better understand how Copilot usage can impact productivity directly, we conducted a survey with users about their perceived productivity and compared it with directly measurable usage data. (For a full description and findings of this study, see Ziegler, et al., "Productivity assessment of neural code completion."[7])

Survey participants were those users in the technical

preview who opted in to receive communications. Of the 17,420 surveys sent, 2,047 responses were received between February 10, 2022, and March 12, 2022. The survey included questions along several dimensions of productivity based on the SPACE framework.[2]

Analysis of their responses showed that users' perceived productivity correlated to 11 usage metrics, including persistence rate (percentage of accepted completions that remained unchanged after various intervals); contribution speed (number of characters in accepted completions per hour); acceptance rate (percentage of shown completions accepted by the user); and volume (total number of completions shown to the user). The paper includes a full list of the adapted metrics used. Across all analyses, acceptance rate has the highest positive correlation to aggregate perceived productivity ($\rho = 0.24$, $P < 0.0001$)—higher than persistence measures.[7] This finding implies that users perceive themselves to be more productive when accepting suggestions from Copilot, even if they have to edit the suggestion. That said, this presents opportunities to explore how suggestions that help users think and tinker may be more valuable to them than ones that save them typing time.[6]

DISCUSSION

Copilot is one of the first widely used developer tools powered by AI models, offering a notable shift over previous methods like genetic programming.[3] Over the next five years, however, AI-powered tools likely will be helping developers in many diverse tasks. For example, such models may be used to improve code review, directing

reviewers to parts of a change where review is most needed or even directly providing feedback on changes. Models such as Codex may suggest fixes for defects in code, build failures, or failing tests. These models are able to *write* tests automatically, helping to improve code quality and downstream reliability of distributed systems.

AI models could help refactor large, complex code bases, making them easier to read and maintain. Code comments or documentation may be automatically generated using these models. In short, any developer task that requires interacting or reasoning about code in any way can likely be aided by AI. The challenge will come in creating the right user experience such that the developer is helped more than hindered.

This study of Copilot shows that developers spend more time reviewing code (as suggested from Copilot or similar tools) than actually writing code. As AI-powered tools are integrated into more software development tasks, developer roles will shift so that more time is spent assessing suggestions related to the task than doing the task itself (e.g., instead of fixing a bug directly, a developer will assess recommendations of bug fixes).

In the context of Copilot, there is a shift from *writing* code to *understanding* code. An underlying assumption is that this way of working—looking over recommendations from tools—is more efficient than doing the task directly without help. These initial user studies indicate that this is true, but this assumption may not always hold for varying contexts and tasks. Finding ways to help developers understand and assess code—and the context within which that code executes and interacts—will be important. This

naturally leads to the need to understand the dynamics between developers and AI-powered tools.

An increasingly important topic of consideration is the trust that developers have in AI-powered tools like Copilot. The success of any new development tool is tied to how much a developer trusts that the tool is doing the "right thing." What factors are developers finding important to build that trust? How does trust get reconstructed after AI-powered developer tools perform in an unexpected manner? For example, if code suggested by Copilot introduces security vulnerabilities or performance bottlenecks, its use will rapidly decline.

"Traditional" tools such as compilers or version control systems are largely deterministic and predictable. When problems occur, developers can examine and even modify the source code to understand any unexpected behavior. That is simply not the case with AI-powered tools. In fact, AI deep-learning models are probabilistic and much more opaque. Further research is needed to better understand how tools leveraging these AI models can be designed to foster developer trust, leading to a measurable positive impact with developers.

Finally, it will become important to track AI-generated code throughout the software development life cycle, as this will help answer important questions: Does AI-generated code lead to fewer (or more?) build breaks, test failures, or even post-release defects? Should such code have more or less scrutiny during code review? What proportion of shipping code comes from tools such as Copilot?

The answers to these questions are important to all

stakeholders of a software organization, but answering them requires knowing where each line of code comes from. Unfortunately, these answers are unknown right now: The provenance of generated code doesn't live past a single development session in an IDE. There is no way to know which code checked into a git repository comes from an AI tool. Developing provenance tools that can track generated code end to end from IDE to deployment will be critical for software organizations to make informed decisions about when, where, and how to incorporate AI-powered tools into their development.

CLOSING

This research on Copilot provided early insight into how AI-powered tools are making an entrance into the software-development process. Likewise, these studies have also presented new research questions that warrant further investigations for AI-powered tools overall[1]. We hope that these early findings inspire readers to consider what this can mean for the nature of collaboration for their work in the future and its potential impact.

HOT OFF THE PRESS: RESEARCH POINTERS

AI pair programming is a relatively new area that practitioners and researchers are actively exploring. If you are interested in reading more about the topic, there are some recent papers and studies, organized in two main categories: (1) how programmers are using Copilot; and (2) the effectiveness of Copilot. (Note that Copilot was the first AI pair programmer to be

released and was the tool with the largest distribution, so most research is done with this tool.)

How Programmers are Using Copilot

➡ Vaithilingam, P., Zhang, T., Glassman, E. 2022. *Expectation vs. experience: evaluating the usability of code generation tools powered by large language models. Extended Abstracts of the 2022 Conference on Human Factors in Computing Systems. Article no. 332, 1-7;* https://dl.acm.org/doi/10.1145/3491101.3519665*; https://tianyi-zhang.github.io/files/chi2022-lbw-copilot.pdf.*

➡ *Barke, S., James, M. B., Polikarpova, N. 2022. Grounded Copilot: how programmers interact with code-generating models. Arxiv.org; https://arxiv.org/abs/2206.15000.*

Vaithilingam, et al. share insights on how 24 students use Copiloft for three programming tasks and its impact on task completion time and success rate. Barke, et al. report on a grounded theory analysis of how 20 programmers interacted with Copilot. They observed modes: In acceleration mode, a programmer uses Copilot to get to the code faster; in exploration mode, a programmer uses Copilot to explore the options.

Effectiveness of Copilot

➡ Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., Jiang, Z. M. 2022. *GitHub Copilot AI pair programmer: asset or liability? Arxiv.org; https://arxiv.org/abs/2206.15331.*

➡ Nguyen, N., Nadi, S. 2022. *An empirical evaluation of GitHub Copilot's code suggestions. In* Proceedings

of the IEEE/ACM 19th International Conference on Mining Software Repositories, 1-5; https://dl.acm.org/doi/10.1145/3524842.3528470.

➡ Imai, S. 2022. *Is GitHub Copilot a substitute for human pair-programming? An empirical study. In* Proceedings of the IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 319-321; *https://dl.acm.org/doi/abs/10.1145/3510454.3522684.*

➡ Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R. 2022. *Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions.* IEEE Symposium on Security and Privacy, 754-768; *https://www.computer.org/csdl/proceedings-article/sp/2022/131600a980/1FlQxERjKCs.*

➡ Asare, O., Nagappan, M., Asokan, N. 2022. *Is GitHub's Copilot as bad as humans at introducing vulnerabilities in code? Arxiv.org; https://arxiv.org/abs/2204.04741.*

➡ Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalan, G., Aftandilian, E. 2022. *Productivity assessment of neural code completion. In* Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, 21-29; *https://dl.acm.org/doi/10.1145/3520312.3534864.*

Dakhel, et al. compare Copilot's solution for fundamental algorithmic problems with human-written code by students. Nguyen and Nadi investigate how well Copilot performs on LeetCode questions and compare its performance across programming languages. Imai investigates the effectiveness of Copilot as a substitute for a human pair programmer. Several papers focus on security. Pearce, et al. found that just like human

developers, Copilot can produce vulnerable code in some cases. Asare, et al. are currently working on an empirical study to investigate whether Copilot is as likely as human developers to introduce vulnerabilities. Ziegler, et al. investigate whether developer interactions with GitHub Copilot can predict self-reported productivity and report patterns in the acceptance rates of Copilot suggestions. (A portion of this work is briefly summarized in the main text's section about the large-scale survey.)

Acknowledgments

We thank the Copilot team for great discussions. We also thank our study participants for offering great insights.

References

1. Ernst, N. A., Bavota, G. 2022. AI-driven development is here: Should you worry? *IEEE Software*, *39(2),* 106-110; https://ieeexplore.ieee.org/document/9713901.
2. Forsgren, N., Storey, M. A., Maddila, C., Zimmermann, T., Houck, B., Butler, J. 2021. The SPACE of developer productivity: There's more to it than you think. *queue* 19(1), 20-48; https://queue.acm.org/detail.cfm?id=3454124.
3. Sobania, D., Briesch, M., Rothlauf, F. 2022. Choose your programming copilot: a comparison of the program synthesis performance of GitHub Copilot and genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1019–1027; https://dl.acm.org/doi/10.1145/3512290.3528700.
4. Vaithilingam, P., Tianyi, Z, Glassman, E. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language

Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 1-7. https://doi.org/10.1145/3491101.3519665
5. Williams, L., 2011. Pair programming. In *Making Software: What Really Works, and Why We Believe It*, ed. A. Oram and G. Wilson, 311-328 .O'Reilly Media.
6. Ziegler, A. 2022. Research: How GitHub Copilot helps improve developer productivity. GitHub Blog; https://github.blog/2022-07-14-research-how-github-copilot-helps-improve-developer-productivity/.
7. Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Shawn Simister, S., Sittampalam, G., Aftandilian, E. (2022). Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS 2022)*, 21–29; https://doi.org/10.1145/3520312.3534864.

Christian Bird *is a senior principal researcher in the Software Analysis and Intelligence (SAINTES) group at Microsoft Research. He is primarily interested in the relationship between software design, social dynamics, and processes in large development projects and in developing tools and techniques to help software teams. His work has focused on code review, branching and merging, developer productivity, and applying AI/ML to software engineering tasks. Christian is an ACM Distinguished Scientist. He received his B.S. from Brigham Young University, and his Ph.D. from the University of California, Davis.*

Denae Ford *is a senior researcher at Microsoft Research in the SAINTES group and an affiliate assistant professor in the*

Human Centered Design and Engineering Department at the University of Washington. Her research lies at the intersection of Online Communities and Empirical Software Engineering. Her latest research investigates open source software as a resource for the broader society, trust in AI code-generation tools, and how to empower the next generation of software developers. More information about her latest research can be found on her website: http://denaeford.me/.

Thomas Zimmermann *is a Senior Principal Researcher in the Software Analysis and Intelligence (SAINTES) group at Microsoft Research. His research uses quantitative and qualitative methods to improve software productivity and to investigate and overcome software engineering challenges. He is best known for his work on software analytics and data science in software engineering. He is a Fellow of the ACM and the IEEE and recipient of the 2022 Edward J. McCluskey Technical Achievement Award. For more information, visit his homepage:* http://thomas-zimmermann.com.

Nicole Forsgren *is a partner at Microsoft Research, where she leads Developer Velocity Lab. She is lead author of the Shingo Publication Award-winning book* Accelerate: The Science of Lean Software and DevOps. *Her work on technical practices and development has been published in industry and academic journals and is used to guide organizational transformations around the world. Her current work investigates AI and its role in transforming the software engineering process. For more information, visit her homepage at* https://nicolefv.com.

Eirini Kalliamvakou *is a staff researcher at GitHub Next, where she leads user studies that shape the team's understanding of user needs, and guide the iteration of prototypes. Currently Eirini is looking at AI's transformational impact on how developers create software. Previously at GitHub, Eirini has led the Good Day project, and the 2021 State of the Octoverse report, and has spoken extensively about developer productivity and happiness.*

Travis Lowdermilk *is a principal UX researcher in the Developer Division at Microsoft. He is especially passionate about connecting product teams with their customers to uncover unmet needs and build innovative products. He's had the opportunity to work with product teams from all over the world, sharing his expertise and inspiring product makers to drive customer connection and empathy into the center of their product-making process. Travis is also the co-author of* The Customer-Driven Playbook *and* The Customer-Driven Culture. *For more information about Travis visit:* https://www.travislowdermilk.com.

Idan Gazit *is a senior director of research at GitHub Next, where he leads the Developer Experiences research team. Prior to that, he led the Heroku Data UX team, and is an alumnus of the Django Web Framework core team. He is a hybrid designer-developer and can usually be found geeking out about the Web, data visualization, typography, and color. He lives in the East Bay with his family and surrounds himself with a rotating cast of half-finished projects.*

CONTENTS