Preprints.org

Article

Not peer-reviewed version

Generative AI in Academics

Brittney Schultz *

Posted Date: 12 September 2024

doi: 10.20944/preprints202409.0955.v1

Keywords: generative AI; GitHub Copilot; Education; Computer Science; OOP

Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Generative AI in Academics

Brittney Schultz

University of Wisconsin – Parkside, Kenosha, WI, USA, mitch036@rangers.uwp.edu

Abstract: There is a bifurcation in the academic voice regarding the adoption or rejection of generative AI in the classroom. This division in pedagogy remains ever present as educators struggle to catch up with advancing AI technology and its staunch integration into student's everyday lives. A resounding concern from educators is for the authenticity of student work and the building of critical thinking skills, both of which become threatened by generative AI tools used without proper guidelines. Educators have a right to these concerns, as studies continue to show how generative AI tools successfully pass assessments and how the AI-coauthored work is challenging to detect or differentiate from student-authored work. In the field of Computer Science, the challenges compound as generative AI technology integrates seamlessly into already approved code-generation tools and development environments. Therefore, an informed and thoughtful decision must be made so that student learning can continue to flourish and evolve in a safe and inclusive setting. This research attempts to go beyond introductory course assessments, where a gap in research currently hangs. As curriculum becomes more challenging and abstract, can generative AI continue to pass assessments and be useful educational assistants? Or will these tools create more confusion for students as prompts become more advanced and prove difficult to interpret? During this empirical study, GitHub Copilot is assessed against the assessments of an advanced CS programming course. The findings present a resounding optimism for generative AI tools, with all assessments passed successfully. The study concludes with recommending an integrated approach of introducing and encouraging the use of generative AI. Collaboration with a partnership mindset will produce the best results.

Keywords: generative AI; GitHub Copilot; education; computer science; OOP

I. Introduction

Imagine it's the Fall of 2023 academic year and you teach computer science courses at a local college or university. The first few weeks have gone well, your students appear engaged and are turning in work on time, which brings you much contentment and pride in your teaching efforts. As you begin grading the first week's assignments, you notice some odd things in your students' code. There are comments that begin with conjunctions like "Assuming that…" for parts of the assignment that were required and shouldn't be assumed or guessed at. There is a surplus of helper methods to perform tasks, which seems studious but not altogether common to find in student code. And now, after grading several assignments, you see the same variable and method names, same program flow choices, and same order of code which should rarely if ever happen unless students are copying each other's work or plagiarizing code from online resources.

Feeling deceived and disappointed, you decide to bring the duplicate assignments to the students in question after class the next week. They look at you in a panic, claiming in earnest that they didn't work with each other and, in some cases, don't even know each other. One brave student speaks up and says that they used an Artificial Intelligence generative coding assistant to do their assignment, and the others concede to have done so as well. They hang their heads looking shameful and you're caught unprepared for what the repercussion should be for such an unprecedented situation.

You ask yourself, *should it be considered plagiarism to use a generative AI tool to assist with writing code for a student's assignment?* The act of plagiarism is taking someone else's work and claiming

© 2024 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

ownership of it. But if your student has a suggested plugin installed in the integrated development environment (IDE) that the student was required to download for your course, and they use it to help them write their code, would you view it as theft?

This question and the impact of its answer is what launched this research and exploration into generative AI coding assistance tools. Afterall, without hands-on experience with this technology and knowledge of its pros and cons, how can an instructor make an informed decision? Furthermore, if students find that generative AI coding assistants are encouraged to be used in the workplace, but they were shamed for or banned from using it in the classroom, these kinds of mixed messages will foster distrust, confusion, and may ultimately leave students unprepared for what they'll face after graduation.

The goal of this research is to test out a popular generative AI coding assistant, GitHub Copilot, to see how, if at all, it should be used in academics. With emerging technologies such as generative AI, it's always a struggle to stay informed and pave a safe path for use whether for the public, for students, or even in the home. But in this case, it's hard to imagine that anyone was prepared for this technology to hit mainstream the way that it did.

II. Related Work

Since the Fall of 2022 and the emergence of OpenAI's generative Artificial Intelligence brainchild Chat-GPT, the academic sector has been scrambling to define and enforce the rules of usage, grappling with the security impact, and bracing for the ethical and educational implications of generative AI tools in and out of the classroom. Particularly in the field of computer science, and with a growing list of code-completion AI solutions, there appears to be a precipice that CS educators are standing upon, and it comes with a steep choice: to resist or embrace [1]. There's a path backward that leads to tightened restrictions on plagiarism, a top concern of educators regarding generative AI solutions [1,2], and regresses to perceivably easier to control assessment strategies like handwritten exams and oral presentations [1]. Worries about appropriateness and over-reliance [3,4] of such technology diminishing student learning pushes educators towards this path. The other path cautiously forward integrates such tools into instructional demonstration and assessments, all with a possible new shift in course design – away from *programmer as a creator* and more towards *programmer as an AI code revisor* [3–5]. But how much can we trust the work of our AI pair-programmers? Do we have enough evidence to support the adoption of such technology so completely into CS course curriculum? Or should we treat it as a new programming team member, subject to code-review and job performance evaluation?

Initial perceptions of the use of generative AI tools in academics are mixed among students. Computer Science students are concerned about the misuse of generative AI [6] and a loss of critical thinking and understanding [5,6]. Conversely, students also show improved confidence when using such tools [7] and believe that generative AI can produce a working program without needing to revise the generated code [8]. And they are correct: research has shown that with simple, introductory applications, generative AI tools can complete assignments from first and second semester programming courses with minimal prompting [9,10]. But even with simple assessments, these tools may eventually get stuck repeating the same instructions [5] and can overtime seem to lose understanding of the overall program [11], especially when multiple steps of related reasoning are required [12]. It has been observed that without additional prompting of a programmer, the generated code has a lower chance of being correct [13].

As for our educators, to compete against the use of generative AI tools proves challenging, as plagiarism detection tools have failed to suggest that automated code was plagiarized [6,10]. Productivity tools such as IDEs already provide code completion [4], making it difficult for instructors to differentiate between IDE generated and AI generated code [3]. To make matters worse, generative AI code-completion is showing knowledge of existing academic assignments, able to complete an assignment description and suggest code that has similarities to the assignment after just a few words are prompted [14]. This suggests that whether adoption or rejection of the tool, generative AI is already impacting the classroom. It has been recommended that educators move

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

away from simple assignments [9,11] and instead adopt more advanced material in introductory courses [1,2,9] so that generative AI tools such as GitHub Copilot can be used as a coding assistance tool and not merely a way to complete assignments without student understanding.

Although there has been significant research around the use of generative AI for introductory computer science programming assessments, additional research is still needed to assess the overall credibility and appropriateness of generative AI tools in the classroom, particularly against more advanced programming assessments. A concern, and a motivation behind this research, is that as students learn to rely on generative AI, limitations of the tools and a lack of foundational understanding will impede students from succeeding in post-introductory programming courses. Being able to quickly create working programs without a need to memorize or understand how they work may limit one's ability to draw connections to more abstract programming concepts [15]. It has been argued that a need for foundational understanding and an ability to explain programming choices becomes more important when generative AI tools are used [4,9]. To address the aforementioned concern, this study aims to answer the following research questions:

- How does GitHub Copilot perform as programming assessments increase in difficulty?
- How does the quality of prompt engineering impact the code produced by GitHub Copilot?
- What are some strategies that instructors and students could employ to adopt the use of generative AI tools in the classroom appropriately for student learning?

III. Methodology

A. GitHub Copilot Experiment

To expand upon the work currently done in this field of research, GitHub Copilot was used as an assistant to complete the six programming projects that account for 50% of a students' grade in an advanced Java programming course. This course, offered at a technical college where the author of this research serves as faculty, is designed to expand upon the concepts covered in introductory programming courses, including object-oriented programming (OOP) topics such as abstraction, composition, polymorphism, generics, and even functional programming concepts. To take this course, students would have already completed a recommended minimum of two introductory programming courses [16].

At the time of this study in April 2024, GitHub Copilot (version 1.5.2.5345) had over 10.6 million downloads [17]. Integrated as a plugin in the JetBrains IntelliJ IDEA (version 2023.3.6 Ultimate Edition), Copilot is currently offered free for student use and both Copilot and IntelliJ were used with free student licensing for this research.

The generative AI tool allows for coding assistance in three ways:

- A drawer that, based on your cursor position in a code file, will suggest multiple code implementations for the chosen section of code
- In-editor suggestions with comments used as prompts
- A chat drawer that assists in a conversational approach, similar to OpenAI's ChatGPT

After testing out all three of these modalities, the preferred choice for completing these assignments was to use the chat feature and provide it with a README.md file, a markdown document with the assignment description and requirements taken straight from Canvas, the college LMS. The choice to use the chat feature was twofold. First, instead of one line at a time, Copilot can produce completed class files with a single prompt. Second, the chat feature provides descriptions of the code suggested and allows for follow-up questions, clarification, and proposed improvements.

One goal in this course is to introduce good design practices to students so that in future courses, less detailed assignment requirements can be used while the integrity of design is preserved. Therefore, each of the six assignments include class diagrams and strict requirements that are defined down to instance fields and methods. All of this information was placed into a README document before beginning each experiment and starting a timer.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

Table 1. The six programming assignments.

#	Program Name	Concepts Covered
1	Personality Quiz	Variables, Methods, Console I/O, Loops, Arrays
2	Maze	Classes, Abstraction, Polymorphism, Composition
3	Venn Diagram	Collections, Generics
4	Paint Calculator	File I/O, Try/Catch, Try w/Resource, Exceptions
5	Drivers License	Custom Exceptions, OOP, Dependency Management
6	Dice Game	Functional Programming, Stream API

Each experiment began with an initial prompting for Copilot to review the README file and to suggest how to begin writing the program and with what code specifically to use.

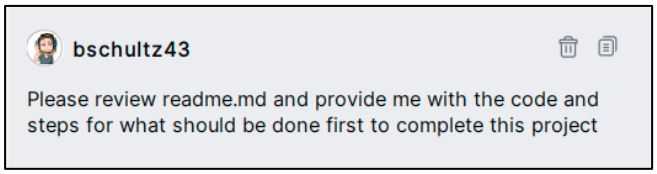

Figure 1. Assignment 2 intial prompt provided to GitHub Copilot chat.

Although these experiments were completed by the author and instructor of this course who knows how the programs should be written, the prompting was kept very simple. The assumptions made for these experiments are that a student would be able to create the code files on their own, chat with Copilot, and paste code into their files. As such, all of the code that was placed in the Java files were copied straight from the Copilot chat screen. No additional code was added by the researcher, unless already provided to the students as part of the assignment.

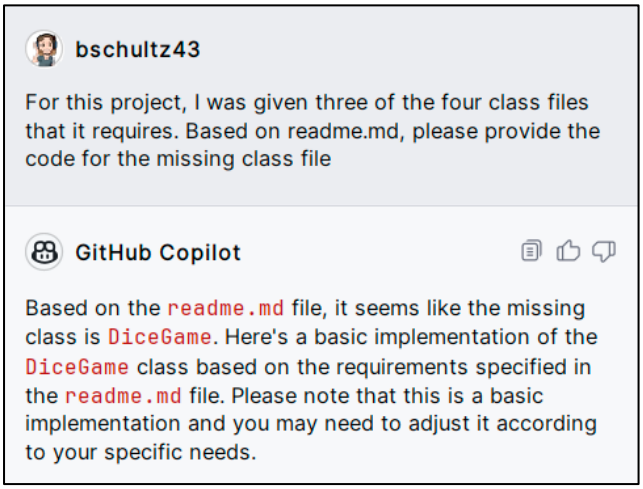

Figure 2. Assignment 6 intial prompt provided to GitHub Copilot chat, where three of the class files already exist.

From there, a series of prompts would continue until there was a runnable program. After code execution, if needed, additional prompts would be given to fix logical errors or missing requirements in the project. Once completed, the time taken was logged and screenshots of the chat log were collected (at the time of this study, the GitHub Copilot plugin for IntelliJ does not allow the chat history to be exported.) Finally, the generated code was assessed against the grading rubric for its corresponding assignment.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

B. Survey Student Perception of Generative AI

In addition to the empirical study, computer science graduate level students were surveyed about their current views on generative AI. The survey asked eleven questions related to generative AI use in academics and in the workplace. Questions were presented primarily as multiple choice questions, with two yes/no questions, and one multiple answer question. There have been recent studies that surveyed students about generative AI [5–8], but as exposure to such tools increases, follow-up surveys can be useful to see overtime how student perceptions change on the subject.

IV. Results

A. GitHub Copilot Experiment Is a Success

After attempting all of the assignments, GitHub Copilot successfully produced code that passed all six advanced programming assignments. There was no instance where a program could not reach 100% requirements met using only Copilot suggested code.

Table 2. The completed assignment data.

#	Program Name	Prompts	Minutes	Files	Completed
1	Personality Quiz	1	5	1	✓
2	Maze	12	40	10	✓
3	Venn Diagram	8	20	2	✓
4	Paint Calculator	5	15	4	✓
5	Drivers License	31	40	11	✓
6	Dice Game	17	40	4	✓

One notably impressive collaboration with Copilot was for assignment 2, the maze, which produced a ten file program in 40 minutes and twelve prompts. This assignment requires three interfaces, a Player class, an abstract Room class, three concrete subclasses of the Room class, a logic class for the game play composed with polymorphic instances of the concrete classes, and a working driver class. Note that within five prompts, there was already a working program with the majority of the requirements met.

It is truly impressive how quickly one can go from reviewing requirements to a working program with the assistance of GitHub Copilot, and there were genuine head to toe awe-inspired moments, both from an instructor perspective and a programmer perspective, during these experiments. On average, approximately twelve prompts were needed to complete each of the six assignments, with an average of 27 minutes spent per assignment.

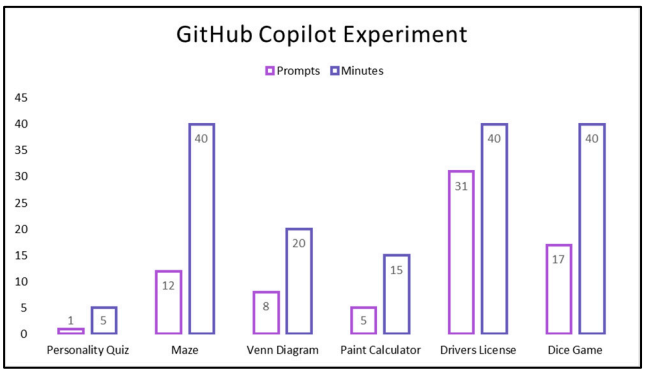

Figure 3. The number of chat prompts and time in minutes required to complete each of the six assignments.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

There is a 78% correlation between time requirement and the number of files an assignment has, suggesting a strong relationship between complexity of design and time needed to complete an assignment. However, assignment 6, the dice game, with only four files and with three of them provided to students, proved to take just as long as assignments 2 and 5, which each require 10 or more files. This shows that even with a small number of files, the speed at which Copilot can complete an assignment in cooperation with the student can be challenging to gauge.

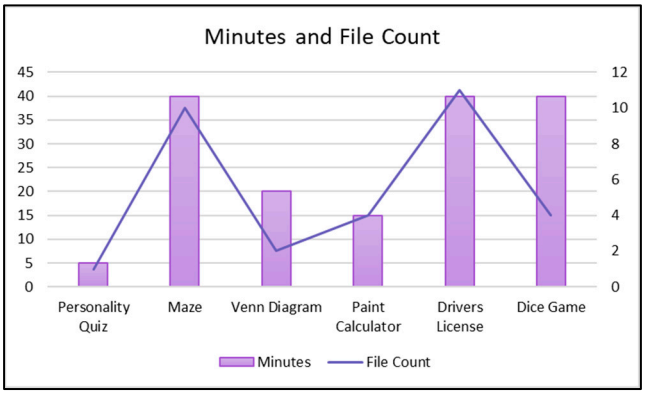

Figure 4. The number of files and time in minutes required to complete each of the six assignments.

Historically, students of this course have needed up to two weeks to complete each of these assignments and more recently, are allowed up to three weeks for assignment 2. So, even at the long end of these experiments, the time savings is enormous. If students can complete their work within an hour of collaboration with GitHub Copilot, imagine the reduction of stress and the opportunity for growth beyond the initial draft of assignments with this gift of time back to students. Furthermore, if these assignments can be completed this quickly, they can be turned into labs during class time, where discussions about design and features can take place and expand well beyond what used to be possible within the confines of a pre-generative AI era.

It is important to note how crucial it is for student learning that Copilot is used *in collaboration* with code creation versus just a means to a swiftly completed assignment. The tool, without enough context, will produce working albeit logically incorrect code very easily. It will use method names that don't exist if the already existing code is not provided as a reference to it, producing code that won't compile. If a student is not reading along with Copilot's suggestions and explanations for those suggestions, the overall experience will be confusing and the student will be left with a program that doesn't work or meet assignment requirements.

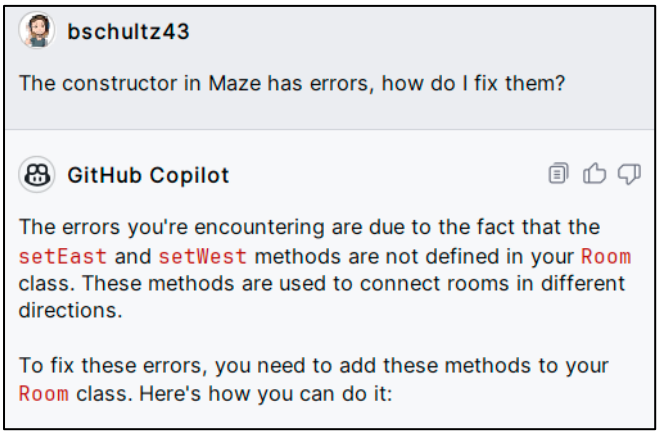

Figure 5. Copilot had suggested code that referenced methods that didn't exist. With additional prompting, the issue is discovered and corrected.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

Real-world topics may be lost on the tool as well. For example, when working on assignment 6, a dice game called Ship, Captain, Crew, there was a working program after five prompts; however, the scoring mechanism was incorrect. It took an additional 8 prompts of various strategies (prompting to review the README, asking clarifying questions, rewording the instructions and providing example dice rolls and describing what the score should be) in order to get code suggested that worked correctly. Additionally, after each of the 8 prompts, Copilot's conversations read very confident, even when the code suggestions are incorrect. Unless the student really knows how the scoring should work, they might not have noticed that the code being suggested was logically incorrect since the program still executes successfully.

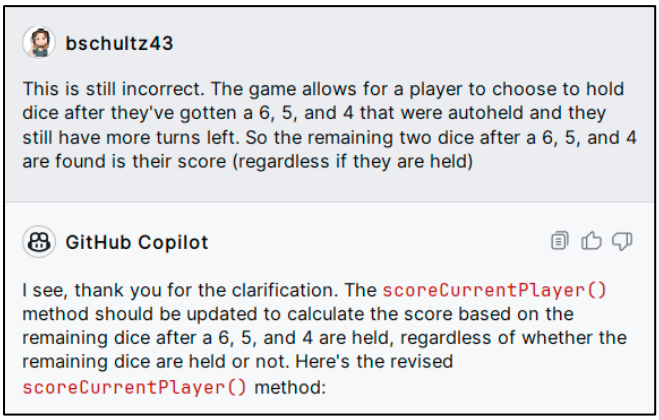

Figure 6. A sample of prompting Copilot to fix a logical issue. Note that the code suggested after this prompt was still logically incorrect even though it seems like it understands the problem.

On the flipside of these concerns is a vast galaxy of potential. When prompting for a concept that was shown in class, Copilot was able to suggest code that aligns with the code that would have already been demonstrated to the students.

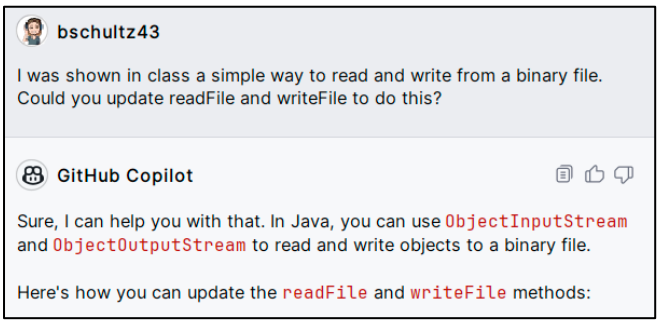

Figure 7. A sample of a prompt from assignment 4 that shows how students can take what they remember from class and ask for code assistance. Note that the code suggested after this prompt exactly matches what the student would have seen in class.

After an assignment is thought to be complete, a student could check if the work produced meets all of the requirements if they provide those requirements as a reference to the tool.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

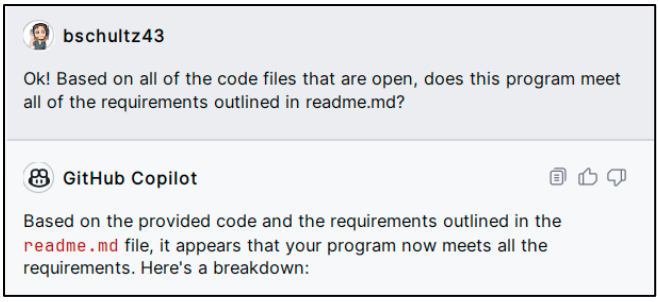

Figure 8. A sample prompt verifying if the program meets all of the requirements.

And if a student questions the code that was suggested, they can ask follow-up questions to improve the code that they have.

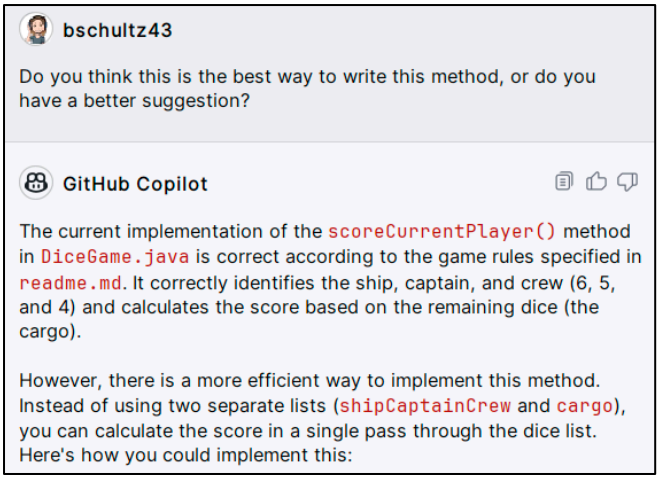

Figure 9. A sample prompt asking if there is a more efficient way to write the code that was originally suggested.

An astute student, with a curious mind and a drive for proficiency would be well-equipped with Copilot at the ready. This collaboration allows for learning outside the classroom, where normally one would have to wait for an instructor to respond to an email or a tutor to meet with when available. It is only when a shortcut in learning is sought from generative AI where wisdom becomes threatened.

Therefore, the strong recommendation from this research is for instructors to teach with these tools from day one, expose the strengths and weaknesses, and expand course curriculum into spaces of richer discussion around code efficiency, refactoring, proper design, identifying logical errors, and brainstorming about the new role programmers are walking into with generative AI as a resource. Creating a safe space to discuss the future of programming may ease the vast concerns plaguing students and instructors alike.

B. Students are Predominantly Concerned

The graduate level computer science student survey was completed by 21 participants. The opinions of these participants show a cautious bend towards adoption of generative AI in academics, but still hold a collective discordant undertone.

To gauge the overall feelings on generative AI, students were allowed to choose multiple answers to describe how they feel on the subject. Of the 23 answers collected, 78% expressed negative feelings about generative AI, with *Concerned* being the top expressed feeling. Only 22% of feelings expressed were optimistic. No students marked that they were confused, indifferent, or unconcerned, suggesting that students are aware of generative AI and have opinions about it.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

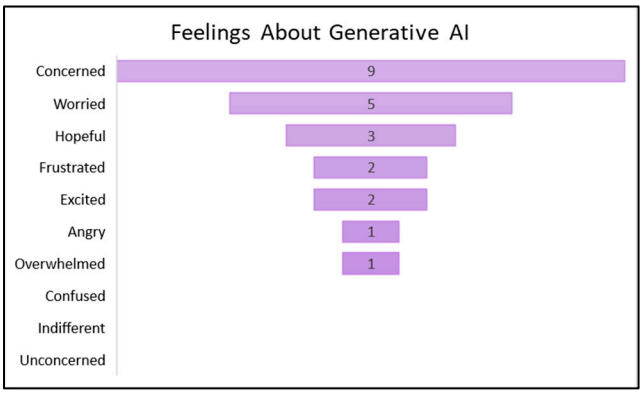

Figure 10. Responses for Overall, how do you feel about generative AI?

Of the 21 responses for *What is your opinion of generative AI tools in academics?*, 71% of respondents believe that these tools should be used, with 62% preferring some clear guidelines. Only 14% were against its use, and another 14% lack a strong opinion on the subject. This suggests an interest in the use of such tools, but also an expressed need for structure.

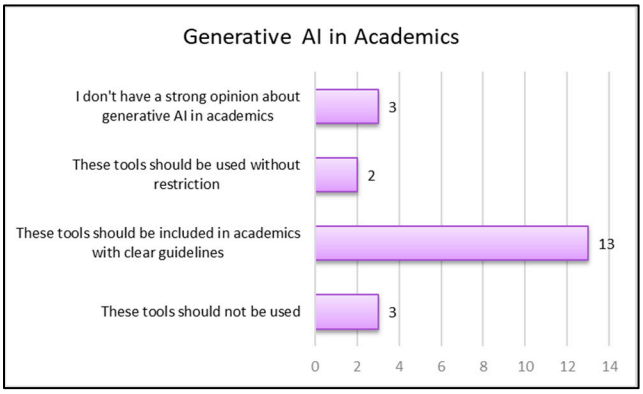

Figure 11. Responses for What is your opinion of generative AI tools in academics?

There was an interesting turn of events when respondents were asked about the use of AI for students and instructors. When asked if students should be allowed to use generative AI tools to complete assigned work, 62% said yes. But when asked if instructors should be allowed to use generative AI tools to grade student work, the same percentage said no. The results suggest that although the majority of students feel that they themselves should have a right to the assistance of generative AI to do their work, their instructors should not when they grade student work.

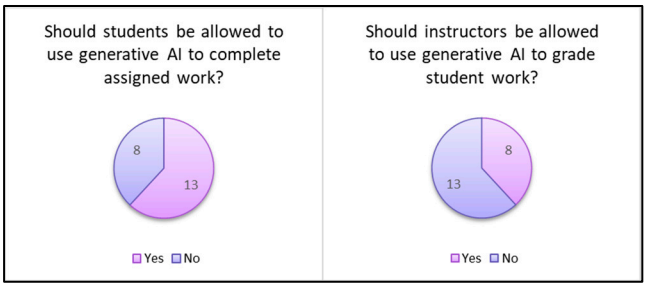

Figure 12. Comparison against who should be allowed to use generative AI.

This survey shows that students still feel in the dark about who should be allowed to use these tools. But with proper guidelines, the darkness can be lifted.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

C. Strategies that Work

The way that a student interacts with generative AI coding assistance tools will shape the overall impact the tool can have on the program and the impression it leaves on the student. For example, the first attempt at completing assignment 6 began without a README file, but instead with referencing the instructor provided driver class (which was broken since it refers to code that doesn't exist yet) and prompting, with comments, to fill in the code that was missing. The Copilot suggestions drawer offered several choices for code to use, which might overwhelm students who do not have enough coding experience to know which is good and which is not. Using this feature of Copilot also doesn't provide explanations for the code suggested as the chat feature does.

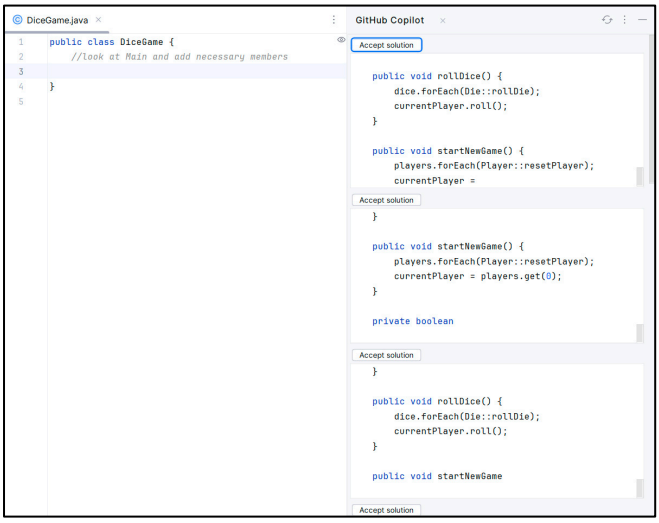

Figure 13. Using the comment feature with Copilot.

It was also noticed that as code generation continued with this format, Copilot would refer to code that didn't exist or rewrite logic that was already present in the program, giving the impression that it was not considering the other parts of the program for the suggestions it was generating. For reference, this first attempt at assignment 6 took over 180 minutes to complete, while it only took 40 minutes when using the chat feature with a README file.

Based on these experiments, here are some suggestions on what worked:

Provide context, either a full requirements document or a summary of what is needed per task and ask for what should be done first. Once you have a raw framework for what's needed, continue with prompting for one task at a time.

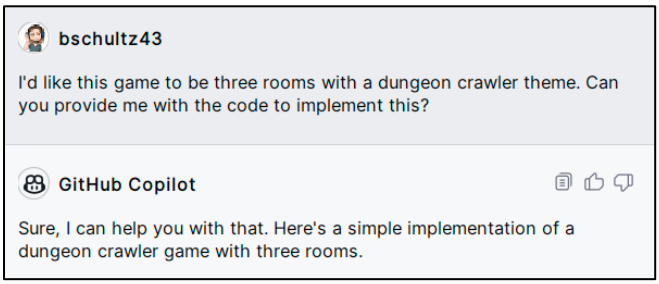

Figure 14. After the first few prompts for assignment 2, the maze, a prompt is given to help build the concrete Room classes, which was left open to students to decide what they should be.

For Copilot to have an awareness of the files in a project, they need to be added as references in the chat using the plus button on the bottom left of the drawer. Copilot does not seem to know about the files in the project unless they are added as references in the chat or they are currently open in the IDE.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

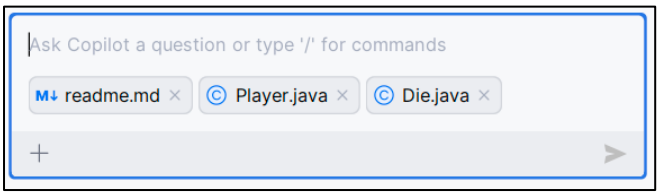

Figure 15. A sample set of references provided to Copilot for assignment 6, the dice game.

Most importantly, ask for clarification. Ask for code to be written in a different way or to match the cadence of the code that already exists. Copilot works like any generative AI technology and picks the best token (in this case, code), one token at a time based on what it was trained is most likely to occur after the previous token [9]. Prompting for additional understanding and revision turns this assistant into a study tool.

V. Discussion and Conclusion

It is clear that, with intentionality and careful prompting, GitHub Copilot can not only be an excellent learning tool, but can without doubt pass advanced programming courses. In partnership with generative AI, code can be explained outside of the classroom in real-time and erroneous syntax and logic can be resolved in a simulated peer-programming setting. If a student is interested in learning and understanding their code, GitHub Copilot will be an excellent tool for them.

Keeping that in mind, if a student is tempted to rush through assignments, generative AI can be confusing, misleading, and even detrimental to learning. To curb this temptation, a completed assignment shouldn't end at code completion. With the swiftness that assignments can be completed with Copilot, there is much room for additional analysis, revision, and discussion that did not exist within the historically necessary time requirements of a college course.

Instructors could build out their assignments into phases: complete the coding assignment with generative AI assistance, review the code that was created, review it again with a partner, discuss findings within a group, make decisions to revise or improve it, and then turn in the project with a recap of findings. Encouraging participation in this way, as a means to assess comprehension in place of stress-inducing strategies like interviews, presentations, and in-person exams, keeps the learning environment within the collaboration theme established with the coding assistant, promoting a culture of engagement and curiosity. Just imagine how freeing it will be for students to not fear judgement for the code they produce, as it becomes coauthored by artificial intelligence, so that they can focus on learning without such insecurities? Students might not initially think that programming and art have much in common until you ask a student to show their code to the class. Writing code is creation, and creation is art.

While generative AI coding assistants continue to be free for students and easy to plug into code editors, students will have equal access and rights to this coding advantage. Instructors are faced with an immense responsibility to create equity in the classroom, so a keen eye on this technology will be important to ensure that students remain in equal standing to complete their work.

There is much room for continued research in computer science with generative AI. The next step would be to test Copilot's performance with concepts such as design patterns, client/server architecture, unit testing, and beyond. As it continues to assess well, GitHub Copilot remains a beneficial tool for students eager to learn.

References

1. S. Lau and P. Guo, "From 'Ban It Till We Understand It' to 'Resistance is Futile': How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot," in *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*, Chicago IL USA: ACM, Aug. 2023, pp. 106–121. https://doi.org/10.1145/3568813.3600138.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 September 2024 doi:10.20944/preprints202409.0955.v1

2. U. Ogurlu and J. Mossholder, "The Perception of ChatGPT among Educators: Preliminary Findings," *RESSAT*, vol. 8, no. 4, pp. 196–215, Dec. 2023. https://doi.org/10.46303/ressat.2023.39.
3. B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, Toronto ON Canada: ACM, Mar. 2023, pp. 500–506. https://doi.org/10.1145/3545945.3569759.
4. P. Denny *et al.*, "Computing Education in the Era of Generative AI," *Commun. ACM*, vol. 67, no. 2, pp. 56–67, Feb. 2024. https://doi.org/10.1145/3624720.
5. C. Bird *et al.*, "Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools," *Queue*, vol. 20, no. 6, pp. 35–57, Dec. 2022. https://doi.org/10.1145/3582083.
6. H. Singh, M.-H. Tayarani-Najaran, and M. Yaqoob, "Exploring Computer Science Students' Perception of ChatGPT in Higher Education: A Descriptive and Correlation Study," *Education Sciences*, vol. 13, no. 9, p. 924, Sep. 2023. https://doi.org/10.3390/educsci13090924.
7. M. Amoozadeh *et al.*, "Trust in Generative AI among Students: An exploratory study," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, Portland OR USA: ACM, Mar. 2024, pp. 67–73. https://doi.org/10.1145/3626252.3630842.
8. O. Petrovska, L. Clift, F. Moller, and R. Pearsall, "Incorporating Generative AI into Software Development Education," in *Proceedings of the 8th Conference on Computing Education Practice*, Durham United Kingdom: ACM, Jan. 2024, pp. 37–40. https://doi.org/10.1145/3633053.3633057.
9. P. Menon, "Exploring GitHub Copilot assistance for working with classes in a programming course," *IIS*, 2023. https://doi.org/10.48009/4_iis_2023_106.
10. B. Puryear and G. Sprint, "GitHub Copilot in the Classroom: Learning to Code with AI Assistance," *Journal of Computing Sciences in Colleges*, Nov. 2022. https://doi.org/10.5555/3575618.3575622.
11. M. Wermelinger, "Using GitHub Copilot to Solve Simple Programming Problems." Accessed: Feb. 10, 2024. [Online]. Available: https://dl.acm.org/doi/epdf/10.1145/3545945.3569830.
12. J. Savelka, A. Agarwal, C. Bogart, Y. Song, and M. Sakr, "Can Generative Pre-trained Transformers (GPT) Pass Assessments in Higher Education Programming Courses?," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, Turku Finland: ACM, Jun. 2023, pp. 117–123. https://doi.org/10.1145/3587102.3588792.
13. B. Yetistiren, I. Ozsoy, and 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*, Singapore: ACM, Nov. 2022, pp. 62–71. https://doi.org/10.1145/3558489.3559072.
14. D. Sobania, M. Briesch, and F. Rothlauf, "Choose Your Programming Copilot: A Comparison of the Program Synthesis Performance of GitHub Copilot and Genetic Programming." arXiv, Nov. 15, 2021. Accessed: Mar. 03, 2024. [Online]. Available: http://arxiv.org/abs/2111.07875.
15. N. C. C. Brown, F. F. J. Hermans, and L. E. Margulieux, "10 Things Software Developers Should Learn about Learning," *Commun. ACM*, vol. 67, no. 1, pp. 78–87, Jan. 2024. https://doi.org/10.1145/3584859.
16. "IT-Web and Software Developer < Waukesha County Technical College," catalog.wctc.edu. https://catalog.wctc.edu/programs/web-and-software-dev/#fulltimeplantext (accessed Apr. 28, 2024).
17. "GitHub Copilot - IntelliJ IDEs Plugin | Marketplace," JetBrains Marketplace. https://plugins.jetbrains.com/plugin/17718-github-copilot (accessed Apr. 28, 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.