Octopilot: An Instructor-Governed AI Coding Environment for Computer Science Education

Frank Stinar[1], Ashton Anderson[2,3], and Jake M. Hofman[3]

No Institute Given

Abstract. Generative AI tools have sparked both excitement and concern in education. Programming assistants like GitHub Copilot are among the earliest and most capable tools to emerge from recent advances in large language models. Yet despite their power, their productive use in classrooms remains unresolved, as they often provide shortcuts that conflict with learning objectives. This challenge stems in part from the difficulty instructors face in adapting and managing these tools for their courses. To address this gap, we built Octopilot, an instructor-controlled AI tool for computer science education that provides both control over AI behavior and visibility into student learning processes. Octopilot is easy to customize and deploy, integrates directly into students' programming environments with education-friendly guardrails, and captures detailed telemetry on individual student progress. We present a pilot deployment in which 11 students engaged in over 1,200 chats with Octopilot across a median of 31 hours, revealing stark differences in how students allocated their time and the types of support they sought. A participatory design study demonstrated strong enthusiasm for tools like Octopilot, while also identifying opportunities for improvement. We show that instructor-controlled AI tools can support constructive classroom use and provide instructors with actionable insights into student learning processes—insights that reveal different aspects of student knowledge and learning gaps than outcome-based assessments alone.

Keywords: computer science education · large language models · learning analytics

1 Introduction

AI and machine learning have been used in education for decades, supporting tasks such as auto-grading and feedback [6, 15], intelligent tutoring systems [18, 1], adaptive learning and testing [11], predicting student performance [21], and learning analytics [20]. Recently, large language models (LLMs) have re-ignited interest in the field, sparking both fears of rampant cheating and deskilling [5], as well as hopes for affordable, personalized instruction [14]. Yet, several years into the widespread availability of LLMs, their impact on education remains unclear. There are many disparate efforts [9, 3, 10], with little convergence on what teachers and students should (or should not) be using—and importantly,

few practical solutions that empower everyday educators to adapt and manage AI for their own courses.

This issue is particularly striking in computer science education, as coding copilots (e.g., Codex [4], GitHub Copilot) were among the earliest deployments of LLMs. These tools, however, were created with practicing software developers in mind, aiming to optimize the speed and efficiency of writing and deploying code. In other words, they were optimized for productivity. Unfortunately, this can conflict with classroom goals and hinder learning when students are just beginning to develop their programming skills [16].

This inspired our work on Octopilot, developed as a turnkey solution for incorporating AI into computer science education. The goals of the project are that it should: be easy for instructors to customize and deploy; be easy for students to install and use; support student learning rather than provide shortcuts; and support instructors in understanding student progress and needs. In other words, instead of being optimized for productivity, Octopilot is optimized for learning.

We designed Octopilot to seamlessly integrate into students' existing development environments. Built as a Visual Studio Code (VSCode) extension, it gives instructors both control and visibility, with convenient but customizable defaults. Inline auto-complete is disabled, while Copilot chat is enabled in an interactive "ask" mode that answers questions but does not automatically implement solutions. The extension also includes a customizable system prompt template that guides instructors in specifying course learning objectives and other preferences to shape student interactions. Additionally, Octopilot logs student activity—including chat prompts and responses, file activity, and focus time—so instructors can better understand students' processes as well as their outcomes.

To test and refine Octopilot, we deployed it in an undergraduate introduction to data science course that one of the authors has taught for the past 12 years. This intensive summer program selects 12 outstanding undergraduates from several hundred applicants and consists of two hours of daily lectures followed by approximately four hours of work each day for four weeks. In an IRB-approved pilot, students were given access to Octopilot for the duration of the course. They consented to share telemetry from the tool and completed pre- and post-surveys on their views of AI in the classroom. At the end of the program, they participated in a half-day participatory design session to provide feedback on the tool, which surfaced strong enthusiasm for Octopilot alongside clear opportunities for improvement. Telemetry and survey data indicated that Octopilot supported learning and revealed differences in how students allocated time and the kinds of support they sought. We used this feedback to further improve the tool and conducted follow-up testing to gather students' opinions on the updates.

In what follows, we first describe related work and how Octopilot builds on it, then discuss the development of the tool, the pilot program, and our results, and conclude with a discussion of future directions. We also provide a working, open source prototype of Octopilot for public use. Together, these contributions illustrate how instructor-controlled AI can support constructive classroom use and provide instructors with actionable, student-specific insights.

2 Related Work

LLMs have generated significant interest in education, with researchers exploring their potential for content generation, individualized tutoring, and intervention design [14, 12, 2]. Early work has shown promise for using LLMs to provide adaptive feedback [6, 15] and enable new forms of interactive learning [3, 10, 17, 19, 7]. Yet, concerns persist about potential negative impacts, including academic dishonesty, and deskilling of students [5]. These concerns are emphasized in computer science education, where LLMs like GitHub Copilot and Codex were designed for productivity rather than learning [4].

The educational use of LLMs for programming has received attention, as these tools can provide shortcuts that may conflict with learning [16]. While some studies explore how students use these tools [9, 8, 13, 19], there remains a gap between research prototypes and solutions that instructors can easily deploy and customize for their own courses. Most existing approaches require technical expertise, lack instructor control, or do not provide visibility into how students are using AI tools. Octopilot addresses this gap by providing a turnkey solution that gives instructors both control over AI behavior and visibility into student interactions, enabling constructive classroom use rather than attempting to prevent AI use entirely.

3 Methods

Our research is split into three parts: (i) developing the Octopilot learning tool, (ii) a pilot study testing Octopilot, and (iii) improving Octopilot through a co-design feedback study.

3.1 Octopilot Design

Octopilot is implemented as a combination of a Javascript/Typescript VSCode extension and a backend database that stores telemetry data (see Figure 1). The VSCode extension integrates directly into students' coding environments, providing safeguarded access to GitHub Copilot's chat functionality and logging student interactions with Copilot and the rest of VSCode's integrated development environment to surface insights for instructors.[1]

As outlined in the introduction, Octopilot gives instructors *control* over how students interact with AI. This control is implemented through a templated Markdown file that instructors can easily edit and customize according to their course objectives and pedagogical goals, and update and re-deploy as they see fit (see Figure 1 in the Supplement for the default prompt template at https://osf.io/9b4sh/overview?view_only=d75c8c35a05b4e11a5de7746b77e3ad5). Additional configuration settings are controlled through a JSON file that disables Copilot's auto-complete feature and set Copilot to "Ask" mode rather

[1] The source code for Octopilot will be made available with the published manuscript.

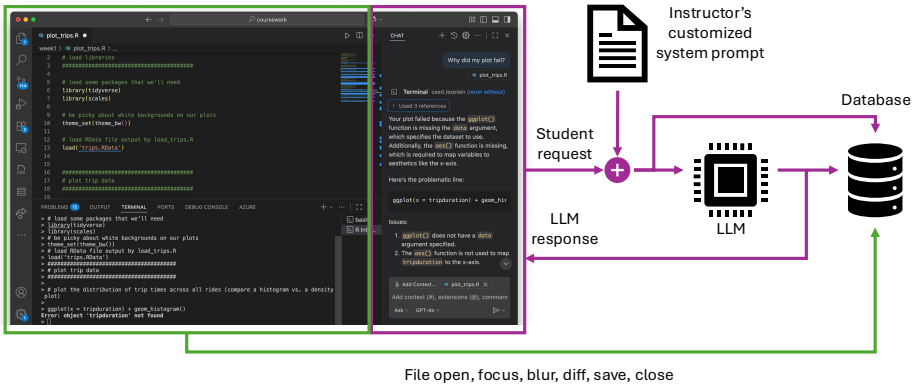

Fig. 1: System architecture overview.

than "Agent" mode, ensuring students can turn to Copilot for support rather than shortcuts. During student use, these guardrails are applied to the generative AI model in VSCode (in our case, GitHub Copilot backed by GPT-4o), shaping responses according to the instructor's specifications.

In addition to instructor control, also Octopilot provides instructors with *visibility* into student learning processes through detailed telemetry collection. Octopilot logs the full sequence of student interactions with assignment files—file open, focus, edit, save, and diffs—as well as window focus and blur events and the sequence of chats with the generative AI model. This surfaces both the amount of time spent on different assignments and the sequence of attention paid to underlying files, capturing progress made in arriving at a solution rather than just the contents of students' final submissions. With these data we can reconstruct individual learning sessions for each student, enabling instructors to understand not just outcomes but the processes by which students arrive at them.

3.2 Pilot data collection

To test and refine Octopilot, we conducted a pilot study within an intensive summer course in data science comprised of 12 undergraduate students. The course lasted four weeks with 20 days of in-person instruction and coursework, for approximately 8 hours each day. Unlike many university courses, all of the course content is intended to be completed in-person with no after-hours assignments. The curriculum does not include formal exams, and instructors provide feedback on assignments only for students' benefit rather than issuing formal grades.

Data collection was approved by an internal ethics review committee, and 11 out of the 12 students consented to their data being collected. Importantly, all students were permitted to use Octopilot throughout the course regardless of whether they consented to their data being collected. As a result, all 12 students used Octopilot, creating a uniform learning environment for all participants.

To improve Octopilot, we conducted a participatory design study with the students at the end of the course. First, during a focus session, the students provided feedback on Octopilot, and we implemented their feedback into an updated prompt for the generative AI. In the following weeks, students rated the updated generative AI responses compared to the responses they received during the course.

4 Results

In the following sections, we present results from our pilot deployment of Octopilot. First, we analyze telemetry to characterize student learning processes, focusing on (i) focus time (time spent editing assignment files), (ii) chat volume (number of prompts sent to the guardrailed LLM), and (iii) chat content (topics of those prompts), and relate these measures to each other and to student outcomes (grades). Second, we summarize post-course student feedback collected via a participatory design session, a survey, and a pairwise rating task comparing alternative system guardrails.

In the analyses below, we highlight three students and five assignments, but all summary statistics reflect the full cohort of 11 students who consented to their data being used for the study. The assignments are (i) Assignment 1: Shell scripting (A1), using Unix tools to count and sort data on the command line; (ii) Assignment 2: Data manipulation (A2), performing statistical calculations in R; (iii) Assignment 3: Data visualization (A3), creating plots in R for exploratory data analysis; (iv) Assignment 4: Literate programming (A4), using RMarkdown to generate an automated research report; and (v) Assignment 5: Research replication (A5), independently reproducing figures from a peer-reviewed academic publication.

4.1 Student Focus

We begin by characterizing overall student activity using two process metrics: focus time and chat volume. Focus time for a given file is the total time that file is the focused tab in the editor, with VSCode as the focused window (many files may be simultaneously open in different tabs, but only one can be in focus at a time; conversely, no file is in focus when the VSCode window is blurred or the machine is idle/off). Chat volume is the number of prompts a student sent to the guardrailed LLM. Across the course, students had a median total focus time just above 30 hours and a median chat volume just under 150 prompts.

Figure 2(a) relates total focus time to chat volume across students. Students span all four quadrants defined by the medians: some combined high focus time with high chat use (frequent consultation while working), others had low focus and low chat use, and notably some showed high chat volume with modest focus time while others worked for long periods with relatively little chat activity. These patterns underscore substantial diversity in how students engaged with the same course materials and support tool. Despite this diversity, these process

Fig. 2: Student engagement patterns. (a) Activity by student, comparing total focus time to number of chat queries. Dashed lines show medians and numbers indicate students' average grade across assignments. (b) Focus time by assignment. Colored points are individual students, open black circles are means, and error bars show one standard error above and below the means.

measures were only weakly related to outcomes (grades)[2]: Pearson correlations were 9% between grades and focus time, and 31% between grades and chat volume, showing that process measures provide largely independent information from outcome measures.

Figure 2(b) shows focus time by assignment, again highlighting large between- and within-assignment variation. More advanced assignments (A5: Research replication) required substantially more time on average than earlier assignments (1.25 hours on average for A1 compared to 5.75 hours for A5), but within each assignment students differed by multiples in time-on-task (e.g., A5 spans roughly 2.5 to 10 hours). Overall, these telemetry traces make visible the tremendous heterogeneity in students' work processes—how long they spend, how much support they seek, and where they allocate attention—that would be difficult to infer from final submissions or grades alone.

Turning to Figure 3, we see that individual-level focus trajectories can yield additional insights for instructors. For each student, the figure shows the sequence of assignment files in focus over time, with gaps corresponding to periods when VSCode was not the active window. For instance, looking at A3 (Data Visualization) and A4 (the Literate Programming), we see meaningful differences in how students navigate between tasks. S5 progresses relatively linearly, spending sustained time on A3 (orange) with only brief returns to other files. In contrast,

[2] Grades, which were used only for these analyses, were computed by applying a grading rubric to student solutions using GPT-4o as an LLM-as-judge, with instructor-authored solutions as a reference.

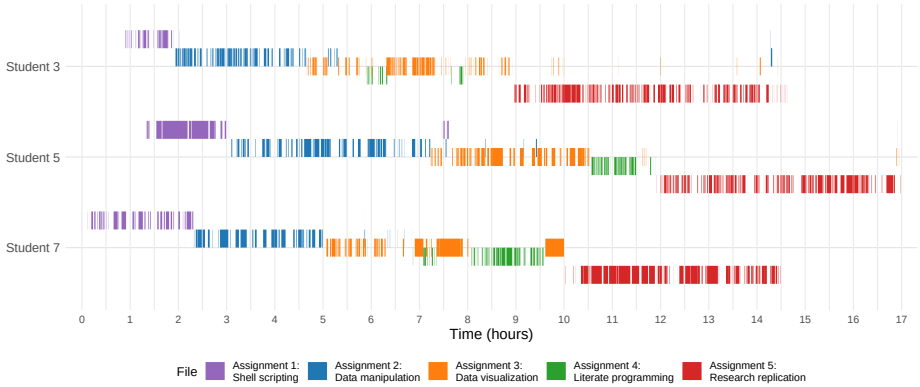

Fig. 3: Working time across the tracked assignments for three students.

S3 and S7 switch more frequently between A3 and A4 (green), which may reflect cross-referencing, iterative trial-and-error, or difficulty getting unstuck.

More generally, patterns in the frequency, timing, and duration of context switching can indicate opportunities for instructor intervention. Rapid, repeated switches clustered within short time spans (e.g., S3) may reflect frustration or confusion, whereas switching that aligns with the course sequence (e.g., S5) may reflect natural task boundaries. Longer switches away from, and back to, the same assignment (e.g., S7) can indicate students moving on before resolving a sticking point.

In this instance, one actionable insight surfaced by these traces was that some "stuck" behavior emerged from challenges encountered along the path to a solution rather than from misunderstandings of the underlying concepts. Although A4, the literate programming assignment, was intended to be conceptually straightforward and students ultimately submitted correct solutions, several encountered dependency issues when installing required RMarkdown packages. This led to repeated exits from VSCode, chats with the LLM to troubleshoot setup, and intermittent returns to the assignment. Without Octopilot's process-level metrics, these intermediate struggles—and how students navigated them—would have been invisible in outcome-based assessments focused solely on final submissions.

4.2 Chat Content

Moving from the volume and timing of activity to its substance, we next analyze what students asked the guardrailed LLM about. To characterize the full corpus of chat prompts, we used a two-step LLM-as-judge process with GPT-4o. First, we inferred a set of concept categories from the full set of prompts; the resulting taxonomy is summarized in Table 1. Second, we applied these categories to tag each individual prompt. For each student, we then counted prompts per category and compared these counts to the typical (median) activity for that

Concept	Description
Stat Concepts	Fundamental principles for drawing conclusions from data
Data Viz	Creating informative plots to communicate data patterns
Data Manip	Transforming, cleaning, and reshaping datasets
Env/System Setup	Installing and configuring software
Data Handling	Working with large and non-standard datasets
Git/Version Control	Tracking changes in code using git
Data Import/Export	Reading data from various file types
RegEx/Text	Pattern-matching to search and clean information from text
Dependencies	Coordinating R and Python packages
Shell/CLI	Interacting with the operating system through commands
Errors/Debugging	Identifying, interpreting, and fixing issues in code

Table 1: LLM-defined concept categories and their descriptions

category across students (the LLM-as-judge prompts are Figures 3 and 4 in the Supplement, respectively).

Figure 4 visualizes these per-student distributions for three students using radar plots, with the dotted line indicating the median number of chats in each concept across all students. These profiles reveal substantial heterogeneity in the kinds of support students sought. For instance, S3 asked an unusually high number of questions about shell scripting, regular expressions, git, and environment setup, suggesting that workflow and tooling issues were a disproportionate source of friction relative to core data science concepts. S5's questions concentrated more heavily on course concepts (e.g., statistics and data manipulation), while remaining close to the typical activity on most workflow-related categories, consistent with using the LLM primarily to deepen conceptual understanding rather than to resolve setup or version control issues. In contrast, S7 engaged at higher-than-typical levels around errors and debugging—suggesting either a higher incidence of difficulty in completing assignments or a greater tendency to use the LLM as a debugging aid.

Overall, concept-level analysis of chat content provides fine-grained visibility into what students struggled with and how they sought support over time. These signals can help instructors refine future lectures, target discussion sections, and direct office hours, as well as identify subgroups of students with similar support needs that may not be apparent from final outcomes alone.

4.3 Improving Octopilot

We sought to improve Octopilot in three ways: (i) evaluating system performance against the intended guardrails, (ii) refining those guardrails through post-course survey feedback and participatory design with students, and (iii) validating whether the resulting changes improved the perceived helpfulness of Octopilot's responses.

We first examined how reliably Octopilot's responses complied with its guardrails. Using an LLM-as-judge approach (prompt is Figure 2 in the Supplement) vali-

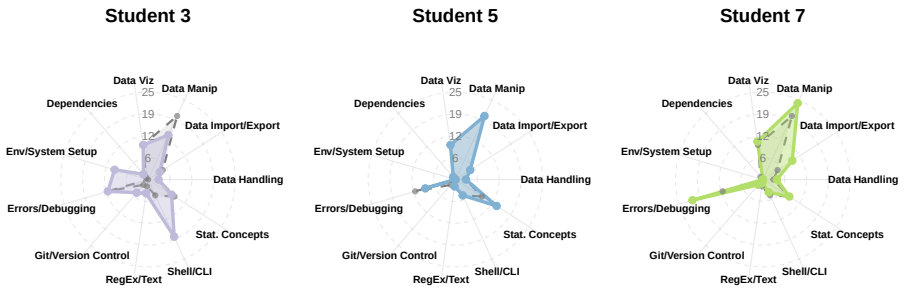

Fig. 4: Concept maps depicting the number of chats Students 3, 5, and 7 had with the LLM over different topics. Colors correspond to Figure 2.

dated by human coders, we classified whether student prompts sought conceptual support or attempted to elicit shortcuts, and whether Octopilot's responses adhered to the guardrails. Three weeks into the course, 1,237 (96.5%) of student prompts sought support rather than shortcuts, while only 44 (3.5%) attempted to obtain direct answers. Octopilot complied with its guardrails for all but two support-seeking prompts and violated guardrails in 26 shortcut-seeking cases. Overall, students overwhelmingly used Octopilot as intended, and the guardrails were largely effective.

After the course concluded, we administered a survey to gather student perspectives on AI use in the classroom. Although some students expressed concern that AI tools could do more harm than good (Figure 6(a)), students nevertheless overwhelmingly agreed that instructor-approved AI tools could enhance their learning and that they preferred to have access to them in future courses.

We then conducted a two-hour participatory design session in which students reflected on their experiences with Octopilot and provided open-ended feedback. A central theme that emerged was that Octopilot was sometimes *too* Socratic—particularly when addressing tooling-related issues such as resolving git conflicts. Because these tasks were not core learning objectives for the course, excessive Socratic prompting in these contexts hindered progress rather than supporting learning. In response, we refined Octopilot's prompt template to distinguish between core conceptual tasks and peripheral tooling challenges. We also updated the system prompt to compare a student's current version of an assignment file against the instructor-provided solution and to provide targeted hints to help students get unstuck without providing too much help (the updated system prompt is Figure 1 in the Supplement).

To evaluate whether these student-informed changes improved the usefulness of Octopilot's responses, we conducted a post-course pairwise rating study. Students were shown their original chat prompts alongside two responses (original

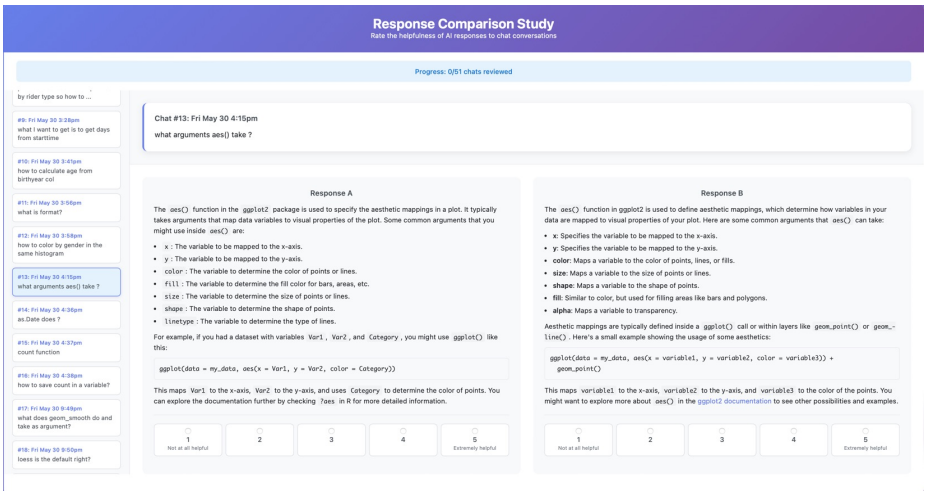

Fig. 5: Pairwise rating interface used to compare original and updated Octopilot responses, presented in random order and anonymized.

vs. updated guardrails), presented in random order and anonymized (blinded to condition) via an online interface (Figure 5). Students rated each response on a five-point scale from "not helpful" to "extremely helpful." As shown in Figure 6(b), the proportion of responses rated as "extremely helpful" increased from 35% under the original guardrails to 50% with the updated, co-designed guardrails, with students rating the refined system as at least as helpful as the original in more than 75% of comparisons.

5 Discussion

Information collected through Octopilot highlights the complexity and diversity of how students engage with course content and generative AI during learning. By combining fine-grained telemetry with analysis of chat content, Octopilot provides insights into learning processes that are difficult to capture through traditional outcome-based assessments alone. These insights have implications for instructional practice, educational AI design, and the use of learning analytics to support students more effectively.

A central contribution of Octopilot is making the distinction between "process" and "outcome" visible to instructors. In most courses, instructors primarily observe outcomes such as assignment submissions and grades, with little insight into how students arrive at those results. Octopilot surfaces intermediate aspects of the learning process, including how long students spend on particular tasks, which files they focus on, and how they engage with generative AI while working. By gaining visibility into these processes, instructors can better distinguish productive engagement with course content and AI tools from behavior that may

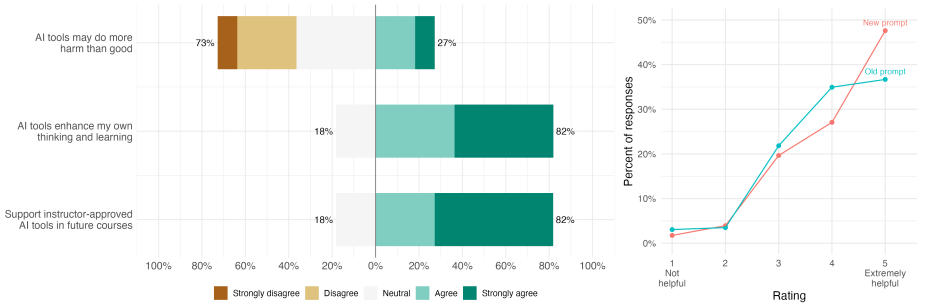

Fig. 6: Sentiment and pairwise rating from post-course surveys. (a) Student sentiment towards AI use for learning after the course. (b) Ratings comparing responses from the original Octopilot prompt to the updated prompt based on the participatory design session with the students.

bypass learning opportunities, enabling more targeted pedagogical interventions. Furthermore, we found minimal correlation between time spent on assignments and the student's grade, indicating that process information provides orthogonal insights to traditional outcome measures.

In addition, the variation we observed in both file focus patterns and chat engagement underscores the individualized nature of learning. Students differed markedly in how they allocated attention across assignments and in the kinds of support they sought from the LLM. Some primarily used the system to address workflow or debugging issues, while others engaged more deeply with core course concepts. These differences suggest that one-size-fits-all approaches to integrating generative AI into instruction may fail to account for meaningful variation in student needs and learning strategies.

Concept-level analysis of chat interactions also provides instructors with visibility into what topics students struggle with during learning—even when those struggles do not surface in final submissions. Students who frequently queried the LLM about core concepts may signal areas where additional instructional emphasis is needed, while those whose interactions focus on tooling or debugging may benefit from different forms of support. Because these signals emerge from students' in-the-moment interactions rather than from post hoc evaluations, they enable earlier and more responsive instructional intervention.

Finally, our findings suggest that students engage with generative AI tools in fundamentally different ways, with important implications for both tool design and instructional practice. Octopilot illustrates how AI tools can be designed to encourage productive learning interactions by default. At the same time, the variation we observed in how students interacted with the system highlights the role of instructors in shaping effective AI use through explicit norms, expectations, and guidance. Together, thoughtful system design and instructional framing can

help align generative AI tools with course learning goals while accommodating diverse student learning strategies.

6 Limitations

This study represents a single pilot deployment conducted within one course and with a small, highly-specific student population. The course structure, instructional goals, and emphasis on data science skills may limit the generalizability of our findings to other educational contexts. In addition, students in this program applied and were selected to participate, and the program specifically targets highly motivated students from under-resourced institutions who would not otherwise have access to this material. As a result, participants may represent a best-case population for constructive engagement with generative AI tools.

The behaviors and outcomes observed in this setting may differ substantially in courses with larger enrollments, less selective admissions, or different incentive structures. In particular, concerns around cheating, gaming the system, or deliberate circumvention of guardrails may be more prominent in other populations. While Octopilot is designed to default students toward constructive use, its current guardrails can be disabled or bypassed, and its effectiveness depends in part on student intent and instructor oversight. Future deployments in broader and more heterogeneous educational settings are necessary to understand how these dynamics change and to evaluate the robustness of instructor-controlled AI tools under less favorable conditions.

7 Conclusion

There are two broad approaches to the use of AI in computer science education. The first is to attempt to ban it—an approach that is likely to be futile, as students are already using AI tools for their coursework and will likely continue to do so. The second is to embrace AI, but to do so in ways that support learning rather than shortcut it.

Octopilot takes the second path by providing an easily configurable environment in which instructors can integrate generative AI into their courses. It gives instructors both control over AI behavior and visibility into student use, recognizing that there is no one-size-fits-all definition of an effective learning aid. Importantly, this approach does not depend on new or more advanced generative AI models, but on thoughtful design and deployment of existing technology in educational contexts.

We argue that enabling broad, constructive use of these technologies in a timely manner is paramount. AI tools are already shaping how students learn and complete coursework, and delaying engagement risks ceding this space to ad hoc, opaque, and pedagogically misaligned use. Octopilot is an open system that democratizes instructors' ability to integrate AI into their courses, supporting iteration and adaptation as the community collectively navigates this rapidly evolving landscape. As norms emerge for how guardrails should be set and evolve

alongside student progress, we hope to create a future in which AI serves as a meaningful complement to learning rather than a substitute for it.

8 Acknowledgements

We thank Mirza Nayeem Ahmed who was the teaching assistant for the course during the Octopilot study. We also thank Chinmay Singh for engineering support in developing the Octopilot teacher dashboard.

References

1. Canfield, W.: ALEKS: A Web-based intelligent tutoring system. Mathematics and Computer Education **35**(2), 152–158 (2001)
2. Cao, C.C., Ding, Z., Lin, J., Hopfgartner, F.: AI Chatbots as Multi-Role Pedagogical Agents: Transforming Engagement in CS Education (Aug 2023). https://doi.org/10.48550/arXiv.2308.03992
3. Chen, E., Huang, R., Chen, H.S., Tseng, Y.H., Li, L.Y.: Gptutor: a chatgpt-powered programming tool for code explanation. In: International conference on artificial intelligence in education. pp. 321–327. Springer (2023)
4. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
5. Hoq, M., Shi, Y., Leinonen, J., Babalola, D., Lynch, C., Price, T., Akram, B.: Detecting ChatGPT-Generated Code Submissions in a CS1 Course Using Machine Learning Models. In: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. pp. 526–532. SIGCSE 2024, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3626252.3630826
6. Jiang, L., Bosch, N.: Short answer scoring with GPT-4. In: Proceedings of the Eleventh ACM Conference on Learning @ Scale. pp. 438–442. L@S '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3657604.3664685
7. Kakar, S., Maiti, P., Taneja, K., Nandula, A., Nguyen, G., Zhao, A., Nandan, V., Goel, A.: Jill Watson: Scaling and Deploying an AI Conversational Agent in Online Classrooms. In: Sifaleras, A., Lin, F. (eds.) Generative Intelligence and Intelligent Tutoring Systems. pp. 78–90. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-63028-6_7
8. Kazemitabaar, M., Hou, X., Henley, A., Ericson, B.J., Weintrop, D., Grossman, T.: How Novices Use LLM-Based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment (Sep 2023). https://doi.org/10.48550/arXiv.2309.14049, http://arxiv.org/abs/2309.14049
9. Liu, R., Zenke, C., Liu, C., Holmes, A., Thornton, P., Malan, D.J.: Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. pp. 750–756. SIGCSE 2024, Association for Computing Machinery, New York, NY, USA (Mar 2024). https://doi.org/10.1145/3626252.3630938

10. Lu, X., Mahesh, A., Shen, Z., Dudley, M., Sano, L., Wang, X.: Exploring LLM-Generated Feedback for Economics Essays: How Teaching Assistants Evaluate and Envision Its Use. In: Cristea, A.I., Walker, E., Lu, Y., Santos, O.C., Isotani, S. (eds.) Artificial Intelligence in Education. pp. 392–406. Springer Nature Switzerland, Cham (2025)
11. Martin, F., Chen, Y., Moore, R.L., Westine, C.D.: Systematic review of adaptive learning research designs, context, strategies, and technologies from 2009 to 2018. Educational Technology Research and Development **68**(4), 1903–1929 (Aug 2020). https://doi.org/10.1007/s11423-020-09793-2
12. Moore, S., Tong, R., Singh, A., Liu, Z., Hu, X., Lu, Y., Liang, J., Cao, C., Khosravi, H., Denny, P., Brooks, C., Stamper, J.: Empowering Education with LLMs - The Next-Gen Interface and Content Generation. In: Wang, N., Rebolledo-Mendez, G., Dimitrova, V., Matsuda, N., Santos, O.C. (eds.) Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky. pp. 32–37. Springer Nature Switzerland, Cham (2023)
13. Nie, A., Chandak, Y., Suzara, M., Ali, M., Woodrow, J., Peng, M., Sahami, M., Brunskill, E., Piech, C.: The GPT Surprise: Offering Large Language Model Chat in a Massive Coding Class Reduced Engagement but Increased Adopters Exam Performances (Apr 2024). https://doi.org/10.48550/arXiv.2407.09975
14. Nye, B.D., Mee, D., Core, M.G.: Generative Large Language Models for Dialog-Based Tutoring: An Early Consideration of Opportunities and Concerns. In: Proceedings of the Workshop on Empowering Education with LLMs. CEUR Workshop Proceedings, vol. 3487, pp. 78–88. CEUR-WS.org (2023)
15. Qinjin Jia, Young, M., Yunkai Xiao, Jialin Cui, Chengyuan Liu, Rashid, P., Gehringer, E.: Insta-Reviewer: A Data-Driven Approach for Generating Instant Feedback on Students' Project Reports (Jul 2022). https://doi.org/10.5281/ZENODO.6853099
16. R. P. Medeiros, G. L. Ramalho, T. P. Falcão: A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. IEEE Transactions on Education **62**(2), 77–90 (May 2019). https://doi.org/10.1109/TE.2018.2864133
17. Rajala, J., Hukkanen, J., Hartikainen, M., Niemelä, P.: "\"Call me Kiran\" – ChatGPT as a Tutoring Chatbot in a Computer Science Course". In: Proceedings of the 26th International Academic Mindtrek Conference. pp. 83–94. Mindtrek '23, Association for Computing Machinery, New York, NY, USA (Nov 2023). https://doi.org/10.1145/3616961.3616974
18. Ritter, S., Anderson, J.R., Koedinger, K.R., Corbett, A.: Cognitive Tutor: Applied research in mathematics education. Psychonomic Bulletin & Review **14**(2), 249–255 (Apr 2007). https://doi.org/10.3758/BF03194060
19. Shah, A., Chernova, A., Tomson, E., Porter, L., Griswold, W.G., Soosai Raj, A.G.: Students' Use of GitHub Copilot for Working with Large Code Bases. In: Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1. pp. 1050–1056. SIGCSETS 2025, Association for Computing Machinery, New York, NY, USA (Feb 2025). https://doi.org/10.1145/3641554.3701800
20. Siemens, G.: Learning Analytics: The Emergence of a Discipline. American Behavioral Scientist **57**(10), 1380–1400 (2013). https://doi.org/10.1177/0002764213498851
21. Yang, T.Y., Brinton, C.G., Joe-Wong, C., Chiang, M.: Behavior-Based Grade Prediction for MOOCs via Time Series Neural Networks.

IEEE Journal of Selected Topics in Signal Processing pp. 1–1 (2017). https://doi.org/10.1109/JSTSP.2017.2700227