# *But \*How\* Do They Use It?: Scaffolding the Introduction of Generative AI Across the SLAC CS Curriculum*

Carolyn Jane Anderson ([carolyn.anderson@wellesley.edu](mailto:carolyn.anderson@wellesley.edu)), Wellesley College and Molly Q Feldman ([mfeldman@oberlin.edu](mailto:mfeldman@oberlin.edu)) Oberlin College

## **Institutional and departmental context**

Location: Wellesley, MA, USA&*Oberlin, OH, USA*

Undergraduate student body size: 2,280 students& *2,550 students (College)*

Degree(s) offered: *Bachelor of Arts*

Department/major name: *Computer Science*

Number of contributing faculty: 19 FTE&*11 FTE*

Number of majors annually: ~75 majors& *~40 majors*

Does the department offer any graduate programs? *No*

Other context (Wellesley): Historically women's college

Other context (Oberlin):*There are an additional 400 students enrolled in the Oberlin Conservatory. Conservatory students can Minor in Computer Science.*

## **Description of Opportunity or Challenge**

Rapid advances in the capabilities of Generative AI (GenAI) tools for programming have provoked much discussion about the future of CS education [15]. On the one hand, many educators are concerned about how over-reliance on generative tools may negatively impact student learning. On the other hand, industry voices claim these tools will level the playing field for programming [1,2]. Unlike at R1 institutions, where students take most coursework within their major, at liberal arts colleges, CS departments must consider how their policies on Generative AI work together with departments across the whole curriculum. Liberal arts colleges aim to produce well-rounded graduates with the ability to think critically and communicate ideas clearly across disciplinary boundaries. It is important for CS departments to consider the impact of Generative AI on these skills as well as programming proficiency. As researchers who study GenAI-user interaction, and SLAC educators who teach across the CS curriculum, we welcome this opportunity to discuss a scaffolded approach to integrating Generative AI into the SLAC CS curriculum.

## **Recommendation or Contribution**

In industry, software professionals work with GenAI tools in multiple interaction modalities. Models can be used to explain code or answer questions about code in natural language, produce documentation, generate code from natural language descriptions, complete existing code, refactor code, generate unit tests, and debug errors. Existing work on student interactions with GenAI for programming has focused on more restricted uses. For instance, in our four-paper exploration of the interactions of CS1 and non-CS students with GenAI programming tools [3,4,5,6], students generate code from natural language descriptions, but we test the code automatically, rather than asking students to verify its correctness themselves. Researchers working with student populations tend to control how students interact with models, not just to simplify their experimental designs, but also out of concern for whether novices will find the full menu of modalities overwhelming.

In fact, prior work suggests that certain ways of working with GenAI for programming can overwhelm students [12]. The emerging consensus regarding GenAI for programming is that such systems are not automatically usable by novice programmers: across a variety of studies with different target populations, non-expert programmers struggle to effectively use GenAI [3,4,12,13,17,20]. Our own research shows that these struggles are not evenly distributed: first-generation CS1 students are less successful at solving tasks with models, while students who come into CS1 with prior experience are more successful [3]. Based on the available evidence, we argue that the key question about Generative AI for programming is how to gradually introduce students to its capabilities without weakening their fundamental skills or exacerbating existing achievement gaps.

The specific liberal arts (SLAC) context presents both opportunities and challenges for GenAI curricular recommendations [23]. For instance, SLAC CS majors take the majority of their classes outside of the department, which creates a variety of opportunities for knowledge transfer. GenAI policies in CS may influence student usage outside of CS, or vice versa. At the same time, CS educators can rely on skills developed in non-CS courses when crafting their CS courses. In addition, SLAC faculty typically do not experience the large-scale enrollment and support concerns that other faculty do, which have motivated the rapid adoption of GenAI in some CS contexts [8,9].

We propose that Generative AI for programming needs to be introduced in a structured and controlled way throughout the curriculum. Students should be exposed to the various uses and interaction modalities in a scaffolded way that supports, rather than supplants, student learning. The adoption of GenAI into CS curricula is currently being discussed in curricula projects [7, 10], but most reports of current adoption are course-specific [8,9,24,25]. Therefore, there are numerous opportunities to consider recommendations at all levels of the educational pipeline.

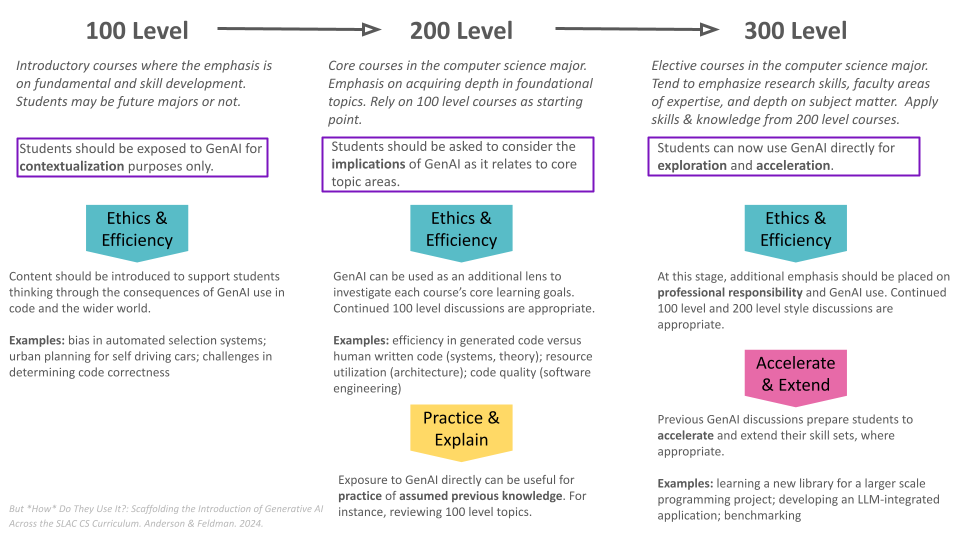
Below (see figure) we provide one possible throughline for scaffolding GenAI usage in SLAC CS departments. We take the perspective that GenAI should be introduced **explicitly and incrementally** in the CS curriculum as it aligns with existing learning goals. Rather than choosing to ignore, fully ban, or fully adopt GenAI in the classroom, we encourage educators to think specifically about when and how these tools can best augment learning. We outline adoption at the “course level” granularity that many SLAC institutions share: 100 level introductory (service) courses, 200 level core courses, and 300 level electives.

In the introductory sequence, our research leads us to believe that the base knowledge provided by 100 level courses is paramount to learn prior to GenAI use, specifically in regards to problem decomposition and problem solving skills [6]. We argue that SLAC students should be exposed to GenAI for **contextualization** purposes only. This allows students to learn computing fundamentals that are crucial to determining if code is correct [12,18], while engaging students with discussions about emerging technology. In the spirit of work on Responsible Computing [22], we envision an *Ethics & Efficiency* theme woven into the introductory sequence that allows students to explore the broader societal impacts of emerging technology. The reflective practice required to understand technology in society aligns with the overall liberal arts educational philosophy [11,23] and allows students to practice discussion and communication skills from non-CS courses.

The 200-level core courses consider domain-specific perspectives on computing. In this context, we consider teaching students about the **implications** of GenAI – they now have enough context to think about GenAI through a precise computing lens. For example, many theory courses emphasize teaching the analysis of runtime. Comparing AI-generated code and student code directly may provide an instructive example of edge cases and program design. At this stage, we also recommend allowing students to use GenAI directly, but sparingly: use should focus on *reviewing* ideas from previous 100-level concepts. We recognize that, in practice, some students enter 200-level courses with incomplete mastery of 100-level topics; use of GenAI for practice or revision may address these gaps [25].

As students work with GenAI in new ways, we envision the Ethics & Efficiency theme continuing throughout the curriculum. As students gain a deeper understanding of how technology works, they can have more nuanced discussions about its ethical impacts, making this topic important to revisit throughout the major.

Finally, the elective level presents the opportunity to introduce GenAI in preparation for students’ likely use post-graduation [16, 19, 21]. GenAI tools can be used here for **exploration** and **acceleration** [14]– students now can discuss and comprehend GenAI’s capabilities. SLAC elective courses also tend to provide specific content foci, while relying on skills, such as writing and communication, from other CS and non-CS courses. Using GenAI to extend projects or learn new skills may be appropriate. At the same time, faculty can engage students in robust discussions about GenAI use both professionally and as they relate to the subject area (e.g., GPU design or data labeling).



**References**

[1] Reuters. (2023, May 29). AI means everyone can now be a programmer, Nvidia chief says. *Reuters*. <https://www.reuters.com/technology/ai-means-everyone-can-now-be-programmer-nvidia-chief-says-2023-05-29/>

[2] Welsh, M. (2022). The end of programming. *Communications of the ACM*, *66*(1), 34-35. <https://dl.acm.org/doi/pdf/10.1145/3570220>

[3] Nguyen, S., Babe, H. M., Zi, Y., Guha, A., Anderson, C. J., & Feldman, M. Q. (2024, May). How Beginning Programmers and Code LLMs (Mis) read Each Other. In Proceedings of the CHI Conference on Human Factors in Computing Systems (pp. 1-26).

<https://dl.acm.org/doi/pdf/10.1145/3613904.3642706>

[4] Feldman, M. Q., & Anderson, C. J. (2024, June). Non-Expert Programmers in the Generative AI Future. In Proceedings of the 3rd Annual Meeting of the Symposium on Human-Computer Interaction for Work (pp. 1-19). <https://dl.acm.org/doi/abs/10.1145/3663384.3663393>

[5] Babe, H. M., Nguyen, S., Zi, Y., Guha, A., Feldman, M. Q., & Anderson, C. J. (2024, August). StudentEval: A Benchmark of Student-Written Prompts for Large Language Models of Code. Findings of the Association for Computational Linguistics. <https://aclanthology.org/2024.findings-acl.501.pdf>

[6] Lucchetti, F., Wu, Z., Guha, A., Feldman, M. Q., & Anderson, C. J. (2024). Substance Beats Style: Why Beginning Students Fail to Code with LLMs. arXiv preprint [arXiv:2410.19792](https://arxiv.org/pdf/2410.19792).

[7] Kumar, A. N., Raj, R. K., Aly, S. G., Anderson, M. D., Becker, B. A., Blumenthal, R. L., Eaton, E., Epstein, S. L., Goldweber, M., Jalote, P., Lea, D., Oudshoorn, M., Pias, M., Reiser, S., Servin, C., Simha, R., Winters, T., & Xiang, Q. Computer Science Curricula 2023. ACM Press, IEEE Computer Society Press and AAAI Press. <https://doi.org/10.1145/3664191>

[8] Vadaparty, A., Zingaro, D., Smith IV, D. H., Padala, M., Alvarado, C., Gorson Benario, J., & Porter, L. (2024). CS1-LLM: Integrating LLMs into CS1 instruction. In Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (pp. 297-303). <https://dl.acm.org/doi/pdf/10.1145/3649217.3653584>

[9] Wang, S., Mitchel, J. C., & Piech, C. (2024). A Large Scale RCT on Effective Error Messages in CS1. In Proc. of the 55th ACM Technical Symposium on Computer Science Education V. 1. <https://dl.acm.org/doi/10.1145/3626252.3630764>

[10] Prather, J., Leinonen, J., Kiesler, N., Benario, J. G., Lau, S., MacNeil, S., Norouzi, N., Opel, S., Pettit, V., Porter, L., Reeves, B.N., Savelka, J., Smith IV, D.H., Strickroth, S. & Zingaro, D. (2024). How Instructors Incorporate Generative AI into Teaching Computing. In Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2 (pp. 771-772). <https://dl.acm.org/doi/abs/10.1145/3649405.3659534>

[11] Becker, J. (2014). What a Liberal Arts and Sciences Education is and is Not. *Liberal Arts and Sciences at the Present Stage: the Experience of the United States and Europe in the Context of Russian Education*  (pg. 12-40). <https://tools.bard.edu/wwwmedia/files/5438940/1/Liberal%20Arts%20and%20Sciences%202014%20Final%20September%202014.pdf>

[12] Prather, J., Reeves, B., Leinonen, J., MacNeil, S., Randrianasolo, A., Becker, B., Kimmel, B., Wright, J., & Briggs, B. (2024). The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1. (pp. 469–486). <https://doi.org/10.1145/3632620.3671116>

[13] Prather, J., Reeves, B., Denny, P., Becker, B., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J., & Santos, E. (2023). “It’s Weird That it Knows What I Want”: Usability and Interactions with Copilot for Novice Programmers. ACM Transactions on Computer-Human Interaction. <https://doi.org/10.1145/3617367>

[14] Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded Copilot: How Programmers Interact with Code-Generating Models. Proc. ACM Program. Lang. 7, OOPSLA1. <https://doi.org/10.1145/3586030>

[15] Lau, S. and Guo, P. (2023). 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 - Volume 1. (pp. 106–121). <https://doi.org/10.1145/3568813.3600138>

[16] Liang, J., Yang, C., & Myers, B. (2024). A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. (pp. 1–13). <https://doi.org/10.1145/3597503.3608128>

[17] Liu, M., Sarkar, A., Negreanu, C., Zorn, B., Williams, J., Toronto, N., & Gordon, A. (2023). “What It Wants Me To Say”: Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. (pp. 1–31). <https://doi.org/10.1145/3544548.3580817>

[18] Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In Proceedings of the Fourth international Workshop on Computing Education Research. (pp. 101–112). <https://doi.org/10.1145/1404520.1404531>

[19] Murali, V., Maddila, C., Ahmad, I., Bolin, M., Cheng, D., Ghorbani, N., Fernandez, R., and Nagappan, N. (2023). CodeCompose: A Large-Scale Industrial Deployment of AI-assisted Code Authoring. <http://arxiv.org/abs/2305.12050>

[20] Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B., & Myers, B. (2024). Using an LLM to Help With Code Understanding. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. (pp. 1–13). <https://doi.org/10.1145/3597503.3639187>

[21] Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. <http://arxiv.org/abs/2302.06590>

[22] Doore, S.A., Rudra, A., Ogunyemi, O., Goetze, T. S., Sahami, M., Cortina, T.J., Bhardwaj, K., & Lee, C. (2023). Teaching Responsible Computing in Context: Models, Practices, and Tools. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2. <https://doi.org/10.1145/3545947.3569613>

[23] Holland-Minkley, A., Barnard, J., Barr, V., Braught, G., Davis, J., Reed, D., Schmitt, K., Tartaro, A., & Teresco, J. D. (2023, March). Computer science curriculum guidelines: A new liberal arts perspective. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (pp. 617-623). <https://dl.acm.org/doi/pdf/10.1145/3545945.3569793>

[24] Lyu, W., Wang, Y., Chung, T., Sun, Y., & Zhang, Y. (2024, July). Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. In Proceedings of the Eleventh ACM Conference on Learning@ Scale (pp. 63-74). <https://dl.acm.org/doi/pdf/10.1145/3657604.3662036>

[25] Liffiton, M., Sheese, B. E., Savelka, J., & Denny, P. (2023, November). Codehelp: Using large language models with guardrails for scalable support in programming classes. In Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (pp. 1-11). <https://dl.acm.org/doi/pdf/10.1145/3631802.3631830>