Less Talk, More Code: Practice-Based Instruction Improves Programming Skill Acquisition

Gillian Gold[1] **(gilliang@andrew.cmu.edu), Jacob Tjaden**[2]**, Paulo F. Carvalho**[1]
[1] Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA 15213
[2] Department of Computer Science, Colby College, Waterville, ME 04901

Abstract

Introductory computer science education often emphasizes watching experts write and explain code. We investigate whether practice-based instruction improves programming skills relative to traditional lectures and which features of practice make it effective. In this preregistered experiment (N = 250), we compared three approaches for an introductory programming lesson: watching a video, code tracing visualizations, and writing code with immediate, AI-generated feedback. Participants who engaged in practice-based instruction performed significantly better than those who watched the video on a novel code-generation test, with the highest performance among learners who practiced writing code. Practice-based instruction also resulted in reduced distraction and increased interest. Whereas video and code tracing conditions led to overconfidence about programming abilities, code writing led to underconfident self-assessments. These findings suggest that active practice, particularly generative practice with feedback, supports learning with advantages in cognitive processing, metacognition, and interest in programming.

Keywords: computer science education; practice-based learning; motivation; metacognition

Introduction

Imagine learning to play piano by watching a concert while an expert explains chord progressions. This approach seems obviously ineffective for learning an instrument; yet, this is how computer programming is often taught. Whether in traditional classrooms or through online tutorials, many novices start by watching experts write and explain code, with little opportunity to practice before being asked to generate programs on their own (Lyon et al., 2023; Rubin, 2013; Stains et al., 2018). This contrast highlights a fundamental problem in computer science (CS) education: if we would not expect someone to acquire complex motor skills without practice, why do we expect students to learn complex computational skills this way?

Active Learning is Effective and Efficient

A large body of research demonstrates that active learning—engaging learners in retrieval, problem solving, or application—improves performance outcomes and reduces failure rates, compared to passive instruction (Carpenter et al., 2022; Freeman et al., 2014; Roediger & Butler, 2011; Roediger & Karpicke, 2006). The "doer effect" similarly shows that practice-based activities are associated with better learning outcomes than passive activities, such as reading or watching lectures (Koedinger et al., 2016; Van Campenhout et al., 2023).

Recent work further suggests that active practice may not only be more effective but also more efficient. For example, in a statistics lesson, learners who engaged in practice alone achieved comparable learning outcomes in less time than those watching an instructional video (Asher et al., 2025).

Despite growing evidence in favor of active learning in STEM domains, CS instructional settings still rely on expert code demonstrations and explanations (Berger et al., 2025; Grissom et al., 2018; Lyon et al., 2023). At the same time, CS courses exhibit high failure and dropout rates (30–50%; Maksimova et al., 2022), with online CS courses showing even higher attrition (over 90%; Jordan, 2014). These patterns raise concerns about whether watching others explain code is effective, or whether early practice is essential for learning to program.

What Type of Practice Matters?

If practice does improve learning in CS, what type of practice is most effective? In CS education, practice formats vary widely: students may trace worked examples, complete scaffolded code, or write programs from scratch (Brusilovsky et al., 2014). Distinguishing among these forms of practice, and the mechanisms through which they support learning, can help clarify what makes practice effective and inform instructional design (Li et al., 2024).

One possibility is that the advantage of practice over lecture instruction is simply from its generative nature, requiring learners to produce responses through effortful retrieval and error correction (Brod, 2021; Bjork, 1994; Bjork & Bjork, 2011; Carpenter, 2009). Related work on productive failure argues that attempting solutions before instruction can promote understanding, particularly when followed by feedback that helps learners identify misconceptions (Kapur, 2008; Kapur, 2015). Alternatively, learners may acquire procedures by mentally rehearsing actions. Processing worked examples by predicting outputs, tracing execution, or explaining reasoning can build internal representations without requiring overt generation (Chi, 2009).

While writing code from scratch imposes higher cognitive load, code tracing reduces this effort by providing a complete solution and asking learners to focus on understanding execution (Xie et al., 2018). Therefore, code tracing may serve as an important intermediate step between passive learning and fully generative practice.

In G. Boleda, I. Dautriche, M. Maldonado, C. Saldana and T. Torrent (Eds.), *Proceedings of the 48th Annual Conference of the Cognitive Science Society*

The Role of Errors in Practice

In addition to active processing, practice can provide opportunities to be wrong. Different practice formats vary in the likelihood of errors and nature of feedback learners receive. These differences may have downstream consequences for not only learning outcomes but also motivation and metacognitive calibration, the ability to accurately judge what one knows (Dunlosky & Metcalfe, 2009; Schraw, 2009).

Practice that involves generation necessarily creates opportunities for errors, and errors themselves may play a role in learning. Making and correcting errors can strengthen memory for correct information (Maraver et al., 2022; Potts et al., 2019). Errors, followed by feedback, can also highlight gaps in understanding that might remain hidden during passive study or certain practice formats (Metcalfe, 2017). On the other hand, errors can be discouraging, reducing motivation and persistence (Eskreis-Winkler & Fishbach, 2019).

The Current Study

In the current study, we explore two central questions: (1) to what extent does active learning improve programming outcomes compared to watching lectures, and (2) what features of practice make it effective. Specifically, we investigate two types of practice to compare the role of actively responding to questions and making errors in supporting learning, and we propose that practice has cognitive processing, metacognitive, and performance advantages compared to watching videos.

We compared three instructional approaches: (a) watching a video, (b) code tracing visualizations, and (c) writing code from scratch with immediate, elaborated feedback. Code writing represents the most open-ended form of programming practice, requiring learners to generate code without scaffolds. Feedback is essential for learning from generative practice. However, providing detailed, real-time feedback on open-ended programming tasks has traditionally been difficult for instructors (Messer et al., 2024; Mirhosseini et al., 2023). To address this limitation, we leveraged AI-generated feedback to offer learners immediate guidance on their code and enabled individualized support without requiring human graders.

Active Processing and Attention. From a cognitive processing perspective, the benefits of practice may arise from active engagement itself, regardless of whether learners construct full solutions. The ICAP framework categorizes student engagement into four modes: Interactive, Constructive, Active, and Passive (Chi & Wylie, 2014). Lecture-based instruction primarily supports passive engagement, whereas code tracing supports active engagement and code writing supports constructive engagement. Both forms of practice require learners to interact with the material, which may help sustain attention and reduce distraction.

If learning gains are primarily driven by active processing, we would expect learners in the lecture condition to exhibit greater distraction than those in either practice condition, but relatively small differences in distraction between code tracing and code writing.

Metacognition and Error-Based Learning. Another goal of effective instruction is to help learners develop accurate self-assessment. Metacognitive calibration—predicted performance compared to actual performance—is critical for self-regulated learning (Dunlosky & Metcalfe, 2009; Schraw, 2009). Learners who are poorly calibrated may stop studying too early, avoid practice they need, or persist with ineffective strategies (Bjork et al., 2013; Foster et al., 2017).

For example, passively watching a lecture can lead to illusions of fluency, when material feels easy to process and learners may infer that they understand it, even when they cannot apply it (Alter & Oppenheimer, 2009; Finn and Tauber 2015; Reber and Greifeneder 2017; Koriat & Bjork, 2005; Schmidt & Bjork, 1992). Further, this can lead to overconfidence and poor self-regulation (Hartwig & Dunlosky, 2012).

By contrast, generative practice with feedback may be especially valuable for calibration because it forces learners to test their understanding (Roediger & Karpicke, 2006). Making errors, and receiving informative feedback about them, may be a key mechanism through which practice improves learning. If this account is correct, learners who engage in more error-prone, open-ended practice (e.g., code writing) should exhibit more accurate calibration than those who engage in less generative practice (e.g., code tracing), and both should outperform lecture-based instruction.

Interest and Confidence. For novices, open-ended programming practice can have high cognitive demands, which may lead to frequent errors and reduce interest. From a utility-value perspective, situational interest is affected by whether the perceived usefulness of an activity outweighs its immediate costs, such as effort, frustration, and the risk of failure (Hecht, Grande, & Harackiewicz, 2021). When errors occur repeatedly, these costs may become salient and ego-threatening, further reducing confidence (Eskreis-Winkler & Fishbach, 2019).

Therefore, practice-based instruction, particularly open-ended code writing, may reduce learners' confidence and interest relative to watching a lecture video. Lecture-based instruction may feel easier, leading to greater ease, enjoyment, and confidence (Deslauriers et al., 2019; Toftness et al., 2018). However, code tracing, which constrains errors and reduces cognitive load, may mitigate some of the motivational costs of practice while still requiring active engagement.

Predictions. Combined, these perspectives suggest that practice-based instruction should improve learning relative to watching videos through a combination of mechanisms. Active processing should reduce distraction and support engagement across practice formats. Moreover, error-driven

practice should further reduce overconfidence, resulting in better calibrated learning that can be sustained.

Consistent with our predictions, practice-based instruction led to better programming performance than lecture-based instruction, and participants who wrote code performed significantly better than those who had code tracing examples. Practice-based instruction was associated with lower distraction and reduced overconfidence. Surprisingly, participants also reported slightly higher interest after practice-based instruction, regardless of increased effort and risk of errors.

Methods

This study was preregistered at https://osf.io/ftwns.

Participants

A total of 266 participants were recruited through Prolific who consented to participate and completed the study. Participants were screened to be ≥18 years old and living in the United States. Using a keystroke tracking tool, we flagged AI use (Asher et al., in press); in total, 16 participants were excluded and replaced.

In the final sample (N = 250), the average age was 39.2 years (SD = 12.7). Of the participants, 45.2% identified as female, 54% as male, and gender information was not available for 0.8% of participants. In terms of race/ethnicity, 64.4% identified as White, 5.6% as Asian, 18.8% as Black, 6.4% as Multiracial, 2.4% as Other, and 2.4% did not have ethnicity information available. In terms of CS background, 65 participants (27.1%) reported having no prior experience, 38 (15.9%) reported completing a high school-level CS class, 65 (27.1%) reported completing a college-level CS course, 18 (7.5%) reported having a degree in a CS-related field, and 50 (20.8%) reported being self-taught. The study took approximately 30 minutes, and participants were paid $6.00 for their participation.

To determine the sample size for this study, we conducted an *a priori* power analysis based on simulated data. We planned to recruit 250 participants (50 for the lecture condition, 100 per practice condition). With this sample and our preregistered contrasts, we estimated >80% power to detect an effect size of d = 0.30 at α = .05 (two-tailed).

Procedure

Participants were randomly assigned to one of three instructional conditions: lecture (N = 50), code tracing (N = 100), or code writing (N = 100). The study had five different sections, which participants completed in sequence: a baseline questionnaire, pretest, a brief programming lesson, an outcome questionnaire, and a final test.

At the beginning of the study, participants were informed that they would learn about CS and completed a baseline questionnaire, rating both their interest and confidence in CS. Next, participants took a pretest with multiple-choice questions adapted from a CS concept inventory. Then, participants proceeded to the CS lesson that covered four learning objectives related to introductory programming concepts: declaring and initializing variables, using arithmetic operators to perform calculations, understanding and applying conditional logic, and using if/else conditional statements to control program flow.

In the lecture condition, participants watched a five-minute video in which the instructor explained these concepts, demonstrated them through short code examples, and summarized how programs execute from top to bottom. In the code tracing visualization condition, participants completed nine worked examples matched to the video's demonstrations. These exercises used PythonTutor visualizations that showed each line of code executing in order, highlighting how variable values changed and how conditional statements affected program flow (Guo, 2013). In the code writing practice condition, participants completed nine short-response programming tasks that closely matched the content and examples presented in the introductory CS lecture. Each task prompted participants to write short snippets of Python code to demonstrate their understanding of key programming concepts, including variable assignment, sequential execution, and conditional logic. Participants received elaborated AI-generated feedback on each submission that identified syntax errors, explained logic issues, and provided the correct solution.

Following the learning session, participants completed an outcome questionnaire assessing their interest in CS, their experience during the learning session, and their confidence and expectations for the final test. Finally, all participants completed a programming test, which included two short-answer generalization questions that required applying learned concepts to novel coding problems.

Materials

The materials used for the questionnaires were previously validated. All questionnaire measures used either five-point Likert-type scales (ranging from "Not at all" to "Extremely") or six-point agreement scales ("Strongly disagree" to "Strongly agree"). Baseline CS assessed with three items (e.g., "How good are you at coding?"; α = .94, M = 2.0, SD = 1.0). Interest in CS was measured pr-instruction using three items (e.g., "How interesting do you find programming?"; α = .92, M = 3.3, SD = 1.0).

Participants completed a brief pretest measuring baseline CS knowledge with three items from the Secondary Computer Science 1 (SCS1) Concept Inventory (Parker, Guzdial, & Tew, 2021). SCS1 is a validated, language-independent assessment tool designed to measure students' understanding of core concepts in introductory CS (Parker, Guzdial & Engleman, 2016). We piloted each question from the inventory with students who had no prior experience to ensure appropriate timing and difficulty.

After instruction, participants self-reported distraction during the learning session, measured using two items (e.g., "I got distracted as I learned during this session"; α = .89, M = 1.7, SD = 0.9). Interest in CS was measured with 4 items and interest in the learning session was measured with 4 items (e.g., "I've enjoyed learning about programming"),

which were then combined (α = .96, M = 4.4, SD = 1.0). Confidence-related outcomes included a three-item measure of confidence in programming abilities (e.g., "How well do you think you would do in a future programming course?"; α = .92, M = 3.6, SD = 1.0). Participants also provided a single-item judgment of learning ("The instruction I just received prepared me well to solve programming problems"; M = 4.2, SD = 1.2). Measures were adapted from Linnenbrink-Garcia et al. (2010), Durik et al. (2015), Hecht et al. (2021), and Asher & Harackiewicz (2024), with judgments of learning drawn from Koriat and Ackerman (2010). Participants also provided estimates of the percentage of questions they expected to answer correctly on the exam on a scale of 0-100% (Hartwig & Dunlosky, 2017) and rated their confidence in the estimate they provided on a scale of 0-10 (Dunlosky et al., 2005).

The final test included two open-ended code generation questions, one focused on variable assignment and one on conditional statements. Responses were evaluated using a scoring rubric designed to capture both syntactic accuracy (e.g., correct use of Python syntax and structure) and logical correctness (e.g., accurate computation and control flow). All responses were independently scored by a human grader and a large language model (LLM) grader trained on the same rubric. Interrater reliability was assessed using a two-way random-effects intraclass correlation coefficient, which indicated excellent agreement between human and LLM ratings (ICC(A, 1) = .93, p < .001).

Results

All analyses were conducted using R Version 4.5.1 (RCoreTeam, 2025). To test our hypotheses, we calculated each participant's scores on the two final test questions, and then fit a series of multiple regression models, each one regressing scores on our two pre-registered orthogonal contrasts: (a) lecture vs. practice (lecture = -2, code tracing = 1, code writing = 1), and (b) code tracing examples vs. code writing with feedback (lecture = 0, code tracing = -1, code writing = 1). We also preregistered that we would include baseline confidence as an interaction term with the condition contrasts. All data and code for this study are openly available at https://osf.io/ns4tf.

Practice Improved Programming Performance

The final test required code generation for two questions: one testing variable assignment and the other testing conditional statements. As seen in Figure 1, participants in both practice conditions outperformed those who viewed the lecture (b = 6.15, t(234) = 4.10, p < .001). Participants who practiced writing code with AI-generated feedback had significantly higher scores than those who only traced worked examples (b = 11.99, t(234) = 6.04, p < .001). Baseline confidence was a significant positive predictor of performance (b = 5.23, t(234) = 2.60, p = .009), indicating that learners who began the session feeling more confident tended to perform better overall. There were no significant interactions between confidence and condition (ps > .16).

These patterns were consistent across both test questions, suggesting that learning generalized across concept types. Participants in both practice-based conditions had higher performance on novel coding questions compared to participants who watched the video, with the greatest gains for those who generated their own code.

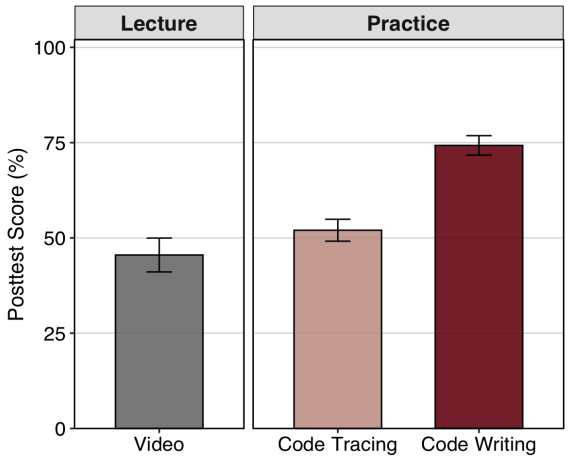

Figure 1: Final Test Performance by Instructional Condition

Practice Reduced Distraction

Participants in the lecture condition reported higher distraction than those in the practice conditions (b = -0.16, t(234) = -2.95, p = .003; see Figure 2). Self-reported distraction did not differ between participants in the code tracing and code writing conditions (b = -0.06, t(234) = -0.78, p = .44). Baseline confidence did not predict distraction (p = .40), and there were no significant interactions between confidence and conditions (ps > .07).

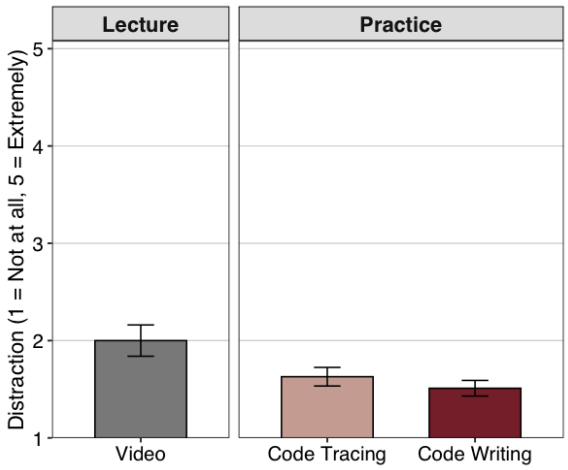

Figure 2: Self-Reported Distraction After Instruction

Code Writing Practice Required More Time

We winsorized time data to remove extreme outliers, defined as values greater than Q3 + 3 × IQR for each instructional timer. There were no significant differences

between instructional time in practice compared to lecture (b = 0.05, t(234) = 0.95, p = .34). Participants in the code writing with feedback condition spent significantly longer in the learning session than those in the code tracing condition (b = 0.32, t(234) = 4.53, p < .001). Baseline confidence did not significantly predict time spent (p = .29), and there were no significant interactions between confidence and conditions (ps > .75).

Practice Reduced Overconfidence on the Final Test

Participants in the lecture and code tracing conditions tended to overestimate their performance, whereas those in the code writing condition underestimated their performance. As seen in Figure 3, participants in practice-based conditions overall exhibited lower overconfidence than those in the lecture condition (b = -7.10, t(234) = -4.26, p < .001). Further, participants in the code writing condition showed significantly lower prediction-performance differences than those in the code tracing condition (b = -15.76, t(234) = -7.15, p < .001), indicating that open-ended generation with elaborated feedback led learners to become more underconfident about their performance. Baseline confidence was a significant positive predictor (b = 8.15, t(234) = 3.65, p < .001), such that learners who began the study more confident also tended to show greater overconfidence overall. There were no significant interactions between baseline confidence and instructional conditions (ps > .17).

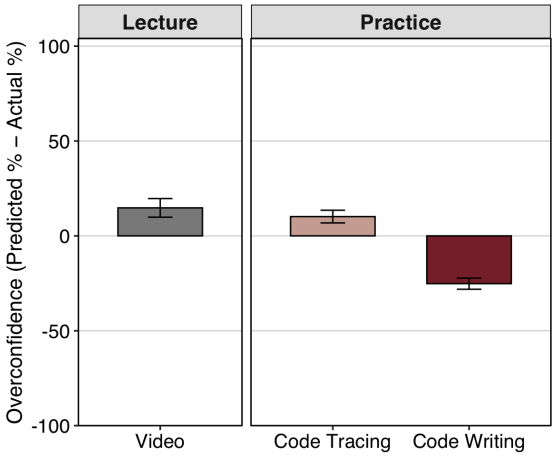

Figure 3: Calibration of Final Test Performance

No Instructional Differences for Confidence or Judgments of Learning

Baseline confidence strongly predicted judgments of learning (b = 0.45, t(234) = 6.78, p < .001). There were no significant differences between practice-based instruction and lecture or between code tracing and code writing practice (ps > .33), and there were no significant interactions (ps > .25). Participants' self-perceived understanding was more influenced by their general confidence than by the instructional format.

Similarly, baseline confidence was a strong positive predictor for post-instruction confidence (b = 0.57, t(234) = 9.27, p < .001). There were no significant differences between practice-based instruction and lecture or between code tracing and code writing practice (ps > .19), and there were no significant interactions (ps > .73).

Practice Supported Interest in Programming

Participants in the practice conditions reported slight but significantly greater post-instruction interest than those in the lecture condition (b = 0.09, t(234) = 2.13, p = .034; see Figure 4). There was no significant difference for interest between participants in practice-based conditions (b = 0.10, t(234) = 1.71, p = .09). We deviated slightly from the preregistered model by including baseline interest as a covariate, given its strong association with post-interest (b = 0.65, t(234) = 12.39, p < .001). There were no significant interactions with instructional conditions (ps > .66).

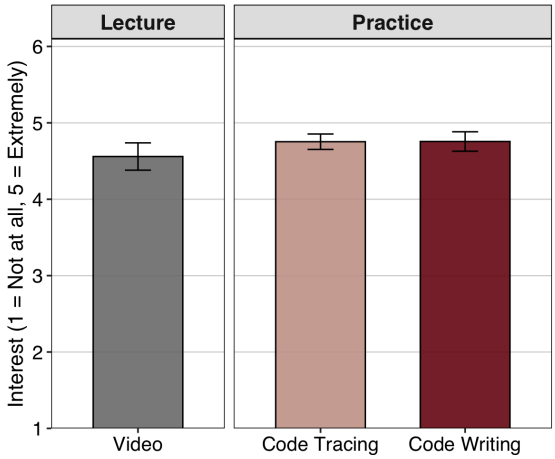

Figure 4: Self-Reported Interest After Instruction

Discussion

The goal of this study was to investigate the effectiveness of practice-based instruction and identify which features of practice contribute to its effectiveness. We found that practice-based instruction resulted in significantly better programming performance on the final test compared to video instruction. Further, participants who wrote code had higher performance than those who were code tracing.

One may think the advantage of code writing practice was from it closely resembling the final test. However, participants who engaged in code tracing, which did not require any code generation, still outperformed those who watched the video. This suggests that the benefits of practice cannot be simply attributed to alignment with assessment format but rather from cognitive engagement, metacognition, and motivation advantages.

Some Types of Practice Reduce Overconfident Self-Assessments

Whereas participants in the lecture and code tracing conditions overestimated their performance, those who practiced writing code underestimated their performance. Overconfidence in these conditions is consistent with theories of fluency-based metacognitive judgments, where ease of processing is misinterpreted as understanding (Alter & Oppenheimer, 2009; Finn and Tauber 2015; Reber and Greifeneder 2017).

In contrast, learners who wrote code became underconfident, despite performing best. It may be the case that, because participants were implicitly expected to fail on initial attempts, errors were normalized as part of the task but also led to a perception of incomplete mastery. Additionally, the AI-generated feedback during code writing identified errors, which made knowledge gaps more salient. Lastly, the task was most similar to the final test, so participants may have developed a more realistic representation of successful performance, leading to more conservative self-evaluations. Together, these factors suggest that open-ended generation with feedback could produce more informative cues about the limits of one's understanding, even if this temporarily suppresses confidence.

Underconfidence may motivate further practice, but it could also discourage persistence if learners interpret difficulty as a sign of low ability (Bouffard & Narciss, 2011; Kubik et al., 2022). In the future, we plan to investigate how changes in calibration interact with learners' mindsets, goals, and subsequent learning choices. Importantly, in this study, underconfidence did not come at the cost of reduced interest, suggesting that self-assessment and sustained motivation need not be in tension.

Errors Do Not Undermine Interest

Despite facing more open-ended tasks with a higher risk of errors, there were no observed negative effects on interest for participants in the code writing condition. Prior work has raised concerns that failure can reduce motivation and lead to disengagement, particularly for novices (Carlson & Fishbach, 2024). However, practice-based conditions led to higher interest compared to participants who watched a video, and there were no differences in interest between the code writing condition and the lower-risk, code tracing condition. Practice-based instruction helped reduce distraction, so active engagement may have increased interest in the topic for future learning.

It is also possible that the immediate and elaborated feedback played a role in sustaining motivation in learning to program (Wang et al., 2019). The motivational impact of errors may depend on instructional context. Errors that are followed by immediate, explanatory feedback can help learners diagnose and resolve misconceptions. In this way, our findings help reconcile tensions between theories of productive failure and concerns about motivation: errors can be productive without necessarily being demotivating when learners are supported in understanding *why* their attempts failed and *how* to improve.

By leveraging AI-generated feedback, this study highlights new opportunities for providing scalable support during open-ended programming practice, addressing a longstanding challenge in CS education (Crow et al., 2020). More broadly, the results suggest that effective instruction should balance cognitive demand, feedback, and opportunities for self-assessment. This may not only help learners perform better but also develop a more accurate understanding of what they know and what they still need to learn. Nonetheless, the study did not manipulate feedback quality or compare AI-generated feedback to human-provided feedback. As a result, it remains unclear which specific properties of the feedback (e.g., elaboration, tone, or adaptivity) were most critical for supporting learning and motivation. We plan to systematically evaluate and vary feedback features to better understand how feedback interacts with errors during practice.

Conclusion

Programming is a generative skill, yet traditional instruction emphasizes passive observation over active practice. Ultimately, our results suggest that low-risk practice formats (e.g., code tracing) can still support learning and engagement without requiring learners to generate code, while high-openness practice (e.g., code writing) can yield the greatest gains in performance and reductions in overconfidence. Critically, encountering errors did not discourage learners when practicing code writing paired with elaborated feedback. AI-generated feedback makes individualized support scalable, enabling learners to engage in authentic programming practice that was previously resource-intensive to deliver.

Theoretically, these results presented here contribute to a better understanding of how learning takes place. Best learning was achieved through a combination of generative practice and reduction in overconfidence. When practice did not reduce overconfidence, it could still improve learning compared to videos, but the effect was smaller. This pattern suggests that learning is an active process of prediction error reduction (Rescorla & Wagner, 1972; Schultz et al., 1997; Metcalfe, 2017), and more robust learning is observed when error reduction is the most effective. Code tracing can allow for some error reduction by students implicitly comparing their expectations with actual code presented, whereas code writing makes it more explicit. Watching videos, as a passive activity, on the other hand, does not require error reduction.

By creating practice environments where errors serve as informative signals rather than discouraging setbacks, we can help learners develop both the technical skills and adaptive mindsets necessary for programming competence. The question is not whether students should practice computational skills, but how we design practice that makes learning effective, motivating, and sustainable.

Acknowledgements

This research was supported by the National Science Foundation under Grant No. 2418656 to Paulo F. Carvalho.

References

Alter, A. L., & Oppenheimer, D. M. (2009). Uniting the Tribes of Fluency to Form a Metacognitive Nation. *Personality and Social Psychology Review*, *13*(3), 219–235. https://doi.org/10.1177/1088868309341564

Asher, M. W., & Harackiewicz, J. M. (2025). Using choice and utility value to promote interest: Stimulating situational interest in a lesson and fostering the development of interest in statistics. *Journal of Educational Psychology*, *117*(4), 647–662. https://doi.org/10.1037/edu0000921

Asher, M. W., Gold, G., Chen, E., & Carvalho, P. F. (2026). Chatbots Are Undermining Crowdsourced Research in the Behavioral Sciences: Detecting AI-Assisted Cheating with a Keystroke-Based Tool. *Advances in Methods and Practices in Psychological Science.* https://doi.org/10.31219/osf.io/a3pzx_v1

Asher, M. W., Sana, F., Koedinger, K. R., & Carvalho, P. F. (2025). Practice with feedback versus lecture: Consequences for learning, efficiency, and motivation. *Journal of Applied Research in Memory and Cognition*. https://doi.org/10.1037/mac0000205

Berger, C., Weintrop, D., & Elmqvist, N. (2025). *"I Feel Like I'm Teaching in a Gladiator Ring": Barriers and Benefits of Live Coding in Classroom Settings* (No. arXiv:2504.02585). arXiv. https://doi.org/10.48550/arXiv.2504.02585

Bjork, E. L., Bjork, R. A., & others. (2011). Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. *Psychology and the Real World: Essays Illustrating Fundamental Contributions to Society*, *2*(59–68), 56–64.

Bjork, R. A. (1994). Memory and Metamemory Considerations in the Training of Human Beings. In J. Metcalfe & A. P. Shimamura (Eds.), *Metacognition* (pp. 185–206). The MIT Press. https://doi.org/10.7551/mitpress/4561.003.0011

Bjork, R. A., Dunlosky, J., & Kornell, N. (2013). Self-Regulated Learning: Beliefs, Techniques, and Illusions. *Annual Review of Psychology*, *64*(1), 417–444. https://doi.org/10.1146/annurev-psych-113011-143823

Bouffard, T., & Narciss, S. (2011). Benefits and risks of positive biases in self-evaluation of academic competence: Introduction. *International Journal of Educational Research*, *50*(4), 205–208. https://doi.org/10.1016/j.ijer.2011.08.001

Brod, G. (2021). Generative Learning: Which Strategies for What Age? *Educational Psychology Review*, *33*(4), 1295–1318. https://doi.org/10.1007/s10648-020-09571-9

Brusilovsky, P., Edwards, S., Kumar, A., Malmi, L., Benotti, L., Buck, D., Ihantola, P., Prince, R., Sirkiä, T., Sosnovsky, S., Urquiza, J., Vihavainen, A., & Wollowski, M. (2014). Increasing Adoption of Smart Learning Content for Computer Science Education. *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, 31–57. https://doi.org/10.1145/2713609.2713611

Carlson, R. W., & Fishbach, A. (2024). Learning from failure. *Motivation Science*, *10*(3), 160–170. https://doi.org/10.1037/mot0000338

Carpenter, S. K. (2009). Cue strength as a moderator of the testing effect: The benefits of elaborative retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *35*(6), 1563–1569. https://doi.org/10.1037/a0017021

Carpenter, S. K., Pan, S. C., & Butler, A. C. (2022). The science of effective learning with spacing and retrieval practice. *Nature Reviews Psychology*, *1*(9), 496–511. https://doi.org/10.1038/s44159-022-00089-1

Chi, M. T. H. (2009). Active‐Constructive‐Interactive: A Conceptual Framework for Differentiating Learning Activities. *Topics in Cognitive Science*, *1*(1), 73–105. https://doi.org/10.1111/j.1756-8765.2008.01005.x

Chi, M. T. H., & Wylie, R. (2014). The ICAP Framework: Linking Cognitive Engagement to Active Learning Outcomes. *Educational Psychologist*, *49*(4), 219–243. https://doi.org/10.1080/00461520.2014.965823

Crow, T., Kirk, D., Luxton-Reilly, A., & Tempero, E. (2020). Teacher perceptions of feedback in high school programming education: A thematic analysis. *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*, 1–6. https://doi.org/10.1145/3421590.3421595

Deslauriers, L., McCarty, L. S., Miller, K., Callaghan, K., & Kestin, G. (2019). Measuring actual learning versus feeling of learning in response to being actively engaged in the classroom. *Proceedings of the National Academy of Sciences*, *116*(39), 19251–19257. https://doi.org/10.1073/pnas.1821936116

Dunlosky, J., & Metcalfe, J. (2009). *Metacognition*. SAGE.

Dunlosky, J., Serra, M. J., Matvey, G., & Rawson, K. A. (2005). Second-Order Judgments About Judgments of Learning. *The Journal of General Psychology*, *132*(4), 335–346. https://doi.org/10.3200/GENP.132.4.335-346

Durik, A. M., Shechter, O. G., Noh, M., Rozek, C. S., & Harackiewicz, J. M. (2015). What if I can't? Success expectancies moderate the effects of utility value information on situational interest and performance. *Motivation and Emotion*, *39*(1), 104–118. https://doi.org/10.1007/s11031-014-9419-0

Eskreis-Winkler, L., & Fishbach, A. (n.d.). *Not Learning From Failure—The Greatest Failure of All*.

Finn, B., & Tauber, S. K. (2015). When Confidence Is Not a Signal of Knowing: How Students' Experiences and Beliefs About Processing Fluency Can Lead to Miscalibrated Confidence. *Educational Psychology Review*, *27*(4), 567–586. https://doi.org/10.1007/s10648-015-9313-7

Foster, N. L., Was, C. A., Dunlosky, J., & Isaacson, R. M. (2017). Even after thirteen class exams, students are still

overconfident: The role of memory for past exam performance in student predictions. *Metacognition and Learning*, *12*(1), 1–19. https://doi.org/10.1007/s11409-016-9158-6

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, *111*(23), 8410–8415. https://doi.org/10.1073/pnas.1319030111

Froyd, J. E., Borrego, M., Cutler, S., Henderson, C., & Prince, M. J. (2013). Estimates of Use of Research-Based Instructional Strategies in Core Electrical or Computer Engineering Courses. *IEEE Transactions on Education*, *56*(4), 393–399. https://doi.org/10.1109/TE.2013.2244602

Grissom, S., Mccauley, R., & Murphy, L. (2018). How Student Centered is the Computer Science Classroom? A Survey of College Faculty. *ACM Transactions on Computing Education*, *18*(1), 1–27. https://doi.org/10.1145/3143200

Guo, P. J. (2013). Online python tutor: Embeddable web-based program visualization for cs education. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 579–584. https://doi.org/10.1145/2445196.2445368

Hartwig, M. K., & Dunlosky, J. (2012). Study strategies of college students: Are self-testing and scheduling related to achievement? *Psychonomic Bulletin & Review*, *19*(1), 126–134. https://doi.org/10.3758/s13423-011-0181-y

Hecht, C. A., Grande, M. R., & Harackiewicz, J. M. (2021). The role of utility value in promoting interest development. *Motivation Science*, *7*(1), 1–20. https://doi.org/10.1037/mot0000182

Jordan, K. (2014). Initial trends in enrolment and completion of massive open online courses. *The International Review of Research in Open and Distributed Learning*, *15*(1). https://doi.org/10.19173/irrodl.v15i1.1651

Kapur, M. (2008). Productive Failure. *Cognition and Instruction*, *26*(3), 379–424. https://doi.org/10.1080/07370000802212669

Kapur, M. (2015). Learning from productive failure. *Learning: Research and Practice*, *1*(1), 51–65. https://doi.org/10.1080/23735082.2015.1002195

Koedinger, K. R., McLaughlin, E. A., Jia, J. Z., & Bier, N. L. (2016). Is the doer effect a causal relationship?: How can we tell and why it's important. *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge*, 388–397. https://doi.org/10.1145/2883851.2883957

Koriat, A., & Ackerman, R. (2010). Metacognition and mindreading: Judgments of learning for Self and Other during self-paced study. *Consciousness and Cognition*, *19*(1), 251–264. https://doi.org/10.1016/j.concog.2009.12.010

Koriat, A., & Bjork, R. A. (2005). Illusions of Competence in Monitoring One's Knowledge During Study. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *31*(2), 187–194. https://doi.org/10.1037/0278-7393.31.2.187

Kubik, V., Jemstedt, A., Eshratabadi, H. M., Schwartz, B. L., & Jönsson, F. U. (2022). The underconfidence-with-practice effect in action memory: The contribution of retrieval practice to metacognitive monitoring. *Metacognition and Learning*, *17*(2), 375–398. https://doi.org/10.1007/s11409-021-09288-2

Li, Y., Chen, M., Hunt, A., Zhang, H., & O'Rourke, E. (2024). Exploring the Interplay of Metacognition, Affect, and Behaviors in an Introductory Computer Science Course for Non-Majors. *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, 27–41. https://doi.org/10.1145/3632620.3671119

Linnenbrink-Garcia, L., Durik, A. M., Conley, A. M., Barron, K. E., Tauer, J. M., Karabenick, S. A., & Harackiewicz, J. M. (2010). Measuring Situational Interest in Academic Domains. *Educational and Psychological Measurement*, *70*(4), 647–671. https://doi.org/10.1177/0013164409355699

Lyon, L. A., Schatz, C., & Green, E. (2023). Experiences of Diverse Introductory Computer Science Students Moving to Online Classes in a Pandemic. *Community College Review*, *51*(4), 593–616. https://doi.org/10.1177/00915521231182112

Maksimova, N., Pentel, A., & Dunajeva, O. (2022). Computer Science Students Early Drop-Out Prediction Using Machine Learning: A Case Study. In M. E. Auer, A. Pester, & D. May (Eds.), *Learning with Technologies and Technologies in Learning* (Vol. 456, pp. 523–549). Springer International Publishing. https://doi.org/10.1007/978-3-031-04286-7_25

Maraver, M. J., Lapa, A., Garcia-Marques, L., Carneiro, P., & Raposo, A. (2022). Can we learn from errors? Retrieval facilitates the correction of false memories for pragmatic inferences. *PLOS ONE*, *17*(8), e0272427. https://doi.org/10.1371/journal.pone.0272427

Messer, M., Brown, N. C. C., Kölling, M., & Shi, M. (2024). Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Transactions on Computing Education*, *24*(1), 1–43. https://doi.org/10.1145/3636515

Metcalfe, J. (2017). Learning from Errors. *Annual Review of Psychology*, *68*(1), 465–489. https://doi.org/10.1146/annurev-psych-010416-044022

Mirhosseini, S., Henley, A. Z., & Parnin, C. (2023). What Is Your Biggest Pain Point?: An Investigation of CS Instructor Obstacles, Workarounds, and Desires. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 291–297. https://doi.org/10.1145/3545945.3569816

Parker, M. C., Guzdial, M., & Engleman, S. (2016). Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment. *Proceedings of the 2016 ACM Conference on International Computing*

Education Research, 93–101. https://doi.org/10.1145/2960310.2960316

Parker, M. C., Guzdial, M., & Tew, A. E. (2021). Uses, Revisions, and the Future of Validated Assessments in Computing Education: A Case Study of the FCS1 and SCS1. *Proceedings of the 17th ACM Conference on International Computing Education Research*, 60–68. https://doi.org/10.1145/3446871.3469744

Potts, R., Davies, G., & Shanks, D. R. (2019). The benefit of generating errors during learning: What is the locus of the effect? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *45*(6), 1023–1041. https://doi.org/10.1037/xlm0000637

R Core Team. (2025). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Reber, R., & Greifeneder, R. (2017). Processing Fluency in Education: How Metacognitive Feelings Shape Learning, Belief Formation, and Affect. *Educational Psychologist*, *52*(2), 84–103. https://doi.org/10.1080/00461520.2016.1258173

Rescorla, R.A. and Wagner, A.R. (1972) A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In: *Black, A.H., Prokasy, W.F., Eds., Classical Conditioning II: Current Research and Theory,* Appleton-Century-Crofts, New York, 64-99.

Roediger, H. L., & Butler, A. C. (2011). The critical role of retrieval practice in long-term retention. *Trends in Cognitive Sciences*, *15*(1), 20–27. https://doi.org/10.1016/j.tics.2010.09.003

Roediger, H. L., & Karpicke, J. D. (2006). The Power of Testing Memory: Basic Research and Implications for Educational Practice. *Perspectives on Psychological Science*, *1*(3), 181–210. https://doi.org/10.1111/j.1745-6916.2006.00012.x

Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 651–656. https://doi.org/10.1145/2445196.2445388

Schmidt, R. A., & Bjork, R. A. (1992). New Conceptualizations of Practice: Common Principles in Three Paradigms Suggest New Concepts for Training. *Psychological Science*, *3*(4), 207–218. https://doi.org/10.1111/j.1467-9280.1992.tb00029.x

Schraw, G. (2009). A conceptual analysis of five measures of metacognitive monitoring. *Metacognition and Learning*, *4*(1), 33–45. https://doi.org/10.1007/s11409-008-9031-3

Schultz, W., Dayan, P., & Montague, P. R. (1997). A Neural Substrate of Prediction and Reward. *Science*, *275*(5306), 1593–1599. https://doi.org/10.1126/science.275.5306.1593

Stains, M., Harshman, J., Barker, M. K., Chasteen, S. V., Cole, R., DeChenne-Peters, S. E., Eagan, M. K., Esson, J. M., Knight, J. K., Laski, F. A., Levis-Fitzgerald, M., Lee, C. J., Lo, S. M., McDonnell, L. M., McKay, T. A., Michelotti, N., Musgrove, A., Palmer, M. S., Plank, K. M., … Young, A. M. (2018). Anatomy of STEM teaching in North American universities. *Science*, *359*(6383), 1468–1470. https://doi.org/10.1126/science.aap8892

Toftness, A. R., Carpenter, S. K., Geller, J., Lauber, S., Johnson, M., & Armstrong, P. I. (2018). Instructor fluency leads to higher confidence in learning, but not better learning. *Metacognition and Learning*, *13*(1), 1–14. https://doi.org/10.1007/s11409-017-9175-0

Van Campenhout, R., Jerome, B., Dittel, J. S., & Johnson, B. G. (2023). The Doer Effect at Scale: Investigating Correlation and Causation Across Seven Courses. *LAK23: 13th International Learning Analytics and Knowledge Conference*, 357–365. https://doi.org/10.1145/3576050.3576103

Wang, Z., Gong, S.-Y., Xu, S., & Hu, X.-E. (2019). Elaborated feedback and learning: Examining cognitive and motivational influences. *Computers & Education*, *136*, 130–140. https://doi.org/10.1016/j.compedu.2019.04.003

Xie, B., Nelson, G. L., & Ko, A. J. (2018). An Explicit Strategy to Scaffold Novice Program Tracing. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 344–349. https://doi.org/10.1145/3159450.3159527