Issues in Information Systems
Volume 26, Issue 1, pp. 458-471, 2025

DOI: https://doi.org/10.48009/1_iis_134

Quantifying shortfalls in students' AI-supported programming practices

Pratibha Menon, *Pennsylvania Western University, menon@pennwest.edu*

Abstract

Generative AI assistants are permeating programming classrooms, yet little is known about whether students adopt sound usage habits. This study surveyed 50 undergraduates in Computer Science / Computer Information Systems courses to examine the frequency of 11 recommended AI-support behaviors, such as planning queries, verifying AI-generated output, and integrating AI with personal code. Substantial shortfalls were observed in critical forethought and verification steps, especially among students using AI for simpler tasks. In contrast, students applying AI to more complex assignments demonstrated stronger behaviors, although certain foundational habits remained weak across the board. The paper proposes five low-overhead scaffolds to help instructors close these usage gaps and improve AI literacy in programming education.

Keywords: AI-assisted programming, self-regulated learning, gap analysis, prompt engineering, computing education.

Introduction

Generative AI tools like ChatGPT and GitHub Copilot have rapidly transformed programming education by providing code generation, algorithm explanations, and real-time assistance, which can personalize and accelerate learning (Lo, 2023; Sun et al., 2024). Studies show that integrating AI into programming courses enhances engagement and performance through instant feedback and tailored support (Zhou, 2023; Yilmaz et al., 2023). LLMs such as ChatGPT adapt to learners' needs, aiding novices in debugging and understanding complex concepts (Kloub & Gupta, 2024; Acosta-Enriquez et al., 2024). However, easy access to AI does not guarantee effective learning; over-reliance may lead to superficial understanding if students bypass essential problem-solving (Prather et al., 2024; Alrayes & Aljohani, 2024). Students with more programming experience or higher self-regulation skills are better able to use AI as a support tool rather than a crutch, integrating AI suggestions into their own problem-solving processes (Prather et al., 2024).

Self-regulated learning (SRL) theory provides a robust lens for understanding effective educational tool use. SRL models, such as Zimmerman's (2002) cyclical phases of forethought, performance, and self-reflection, describe how successful learners plan their approach, monitor progress, deploy strategies, and reflect on outcomes. In programming, SRL might involve planning a solution, seeking relevant information, testing and debugging code, and reflecting on mistakes that align well with using AI assistants as learning aids (Prasad & Sane, 2024; Lan & Zhou, 2025). Recent research suggests that using AI effectively requires metacognitive skills like goal-setting, careful prompt formulation, result verification, and adaptation of

strategies (Knoth et al., 2024; Lan & Zhou, 2025). Students lacking these self-regulatory behaviors may misuse AI (e.g., mindlessly copying code) or fail to obtain its full benefits. For educators, identifying where student practices fall short of best practices when using AI and finding the "gaps" in students' AI-supported programming behaviors is a key challenge.

This study quantifies gaps between recommended and actual AI usage behaviors in programming, showing that novices are more likely to skip critical steps, especially with complex tasks (Mozannar et al., 2024; Al Haque et al., 2024). These insights can inform targeted interventions to foster metacognitive strategies and bridge the gap between current and best practices (Prasad & Sane, 2024; Lan & Zhou, 2025)

Brief Literature Review and Conceptual Model

AI (LLMs) in Programming Education

Integrating AI assistants into coding education is a double-edged sword. On the one hand, AI tutors and code generators offer individualized, adaptive support that traditional teaching often cannot (Roll & Wylie, 2016; Yilmaz & Yilmaz, 2023). Studies show that AI-driven tools can boost engagement and coding performance by providing immediate hints, explanations, and feedback (Zhou, 2023; Al Haque et al., 2024). LLM-based chatbots help simplify complex concepts and guide students through debugging, mainly benefiting novices as they overcome initial hurdles in syntax and logic (Yilmaz & Yilmaz, 2023; Bosch & D'Mello, 2017; Alrayes & Aljohani, 2024).

However, improper use of AI can hinder learning. Students who rely on AI shortcuts may miss more profound understanding and critical skill development (Prather et al., 2024). Over-reliance can erode coding confidence, while advanced learners use AI reflectively, integrating suggestions to enhance their strategies (Tang et al., 2024). Since prior knowledge shapes how students interact with AI, novices need structured guidance to avoid passive use, while experienced learners use AI more collaboratively. Thus, the quality of student-AI interaction is crucial, and educators must foster effective usage behaviors to ensure learning benefits (Rahman & Watanobe, 2023).

SRL and AI Tool Use

Effective AI usage in programming can be conceptualized through SRL theory, which outlines three phases: forethought (planning and goal-setting), performance (monitoring and strategy use), and self-reflection (evaluating and adapting) (Newman, 1994; Zimmerman, 2002). Successful learners employ strategies in each phase to optimize their learning. In AI-assisted programming, these phases map onto distinct student behaviors advocated as best practices by experts (Naamati-Schneider & Alt, 2023; Knoth et al., 2024). For instance, in the forethought phase, a student should analyze the problem and set an approach before using AI-such as outlining requirements or reviewing known solutions rather than immediately querying the tool (Laupichler et al., 2022). In the performance phase, effective strategies include iterative prompting (rephrasing questions for clarity), verifying AI outputs through testing, and integrating AI suggestions with original code (Luckin et al., 2022; Microsoft, 2024). These behaviors align with AI literacy principles emphasizing critical evaluation and ethical use. The conceptual model proposed in this study identifies key behaviors, including problem planning, personal effort, interactive prompting, verification, and content integration, that align with AI literacy principles and are highlighted in recent research (Fenu et al., 2024). These practices help ensure students remain active, critical participants when using AI tools.

In the proposed conceptual model, each behavior has an ideal frequency – essentially, how often a student should engage in it to consider the behavior well incorporated. For critical habits (like testing code or checking requirements), an instructor might argue that "Always" performing them is the goal. Students may

perform these behaviors only occasionally or not at all. We aim to quantify the gap between the ideal (e.g., always) and the actual frequency. This gap analysis approach has been used in educational needs assessments to pinpoint where performance falls short of expectations (Clark & Estes, 2008).

Despite growing interest in AI tutors, research remains limited in identifying the specific self-regulatory practices that enhance AI tool efficacy in computing education. Few studies have quantified student engagement with AI along SRL dimensions or tied those behaviors to contextual factors like task complexity, which this study uniquely addresses.

Research Questions

While theoretical frameworks suggest what students should do with AI tools, there is little empirical data on what students do when left to their devices. Our study fills this gap by systematically measuring students' self-reported AI usage behaviors and comparing them to an ideal benchmark from the above best practices. This study extends prior work by examining an often-hypothesized factor: task complexity. Intuitively, students working on complex programming tasks might be forced to engage in more rigorous practices (e.g., they must break the problem down, test extensively, etc.), or conversely, they might lean more heavily on the AI and skip steps due to the task's difficulty. Recent evidence indicates that without guidance, students will rely on AI in unproductive ways if the course design does not scaffold its role (Darvishi et al., 2024). By comparing high-complexity vs. low-complexity task contexts, we seek to determine whether certain good practices naturally emerge when challenges increase or if additional support is needed for students tackling more complex problems with AI.

The literature suggests that the effective use of AI in programming requires self-regulation and adherence to specific strategies. However, students' adherence to these strategies is uncertain and likely uneven. Findings from literature lead to the research questions, which guide the present study. Based on the foregoing discussion, we pose two research questions:

- **RQ1:** *What is the extent of the shortfall in students' AI-supported programming practices relative to ideal recommended behaviors?* In other words, which specific behaviors do students not perform frequently enough, and how large is the gap for each behavior?
- **RQ2:** *How do these behavioral gaps vary with task complexity?* In particular, do students who engage with AI on complex programming tasks (>50 lines of code) exhibit significantly different (smaller or larger) gaps in these behaviors compared to those who use AI only on simpler tasks?

Answering RQ1 will help identify and quantify student practices' most significant "pain points" (e.g., lack of planning, insufficient testing). Answering RQ2 helps to discern whether complexity is a predictor of better or worse behavior, which can inform whether advanced project settings naturally encourage improved habits or exacerbate certain poor practices. Together, these questions address the diagnosis of current behavior gaps, and the prediction of those gaps based on a key contextual factor.

Methodology

Data Source & Cleaning

To address the research questions, data were drawn from the *Programming with AI Tools* survey (IRB-approved) administered once in Fall 2024 at a public university in Pennsylvania. Invitations went to students in classes from six non-overlapping Computer Science and Computer Information Systems courses, ensuring no individual appeared on more than one roster. Each invitation contained a unique, single-use link that expired after submission; the survey platform also blocked repeated attempts from the

same IP address. Fifty undergraduates (first-year through senior) completed the questionnaire—mean programming experience ≈ 3.5 years, median AI-tool experience ≈ 1 year. Participation was anonymous, voluntary, and limited to one response per student, eliminating duplicate entries. Table 1 in Appendix A lists the eleven behaviour items analysed.

Each behavior was measured with a Likert-style frequency scale: "*In your recent programming assignments, how often did you [behavior]?"* Students responded on a 5-point scale: *Never (1), Rarely (2)* – about 1–2 times per assignment, *Sometimes (3)* – 3–4 times per assignment, *Often (4)* – 5–6 times, and *Always (5)* – 7 or more times per assignment. These anchors (provided in the survey) ensured that students had a concrete sense of what "often" or "always" meant in terms of occurrences per assignment. We treated these responses as ordinal data that can be mapped to numeric values 1–5 for analysis. For instance, a response of "*Always*" was coded as 5 and "*Never*" as 1. Appendix A provides the list of survey questions.

The survey consisted of eleven behavior items aligned with our conceptual model of recommended AI usage (see Table 1), covering key planning and performance-phase practices. These included behaviors such as checking for similar AI-solved problems, writing down needs before prompting, independently understanding the problem, testing and running AI-generated code, rewriting questions, integrating AI and personal code, asking AI to explain unclear code, and switching to manual coding when AI is unhelpful. Each item used a 5-point frequency scale. The survey also asked how often students use AI for complex programming tasks (over 50 lines of code). We used this as a complexity indicator, grouping students who answered *"Often"* or *"Always"* as High-Complexity users and the rest as Low-Complexity users. The survey items in Appendix a shows the three survey pertaining to task complexity.

Content validity of the survey questions was addressed by deriving each item from Zimmerman's SRL constructs (Zimmerman, 2002) and AI-literacy literature, and then having an an industry software engineer review the set to confirm that it fully and appropriately represented the targeted behaviours. Their feedback led to minor revisions in wording but no additions or deletions, supporting the instrument's adequacy in covering the domain of effective AI use in programming.

The survey questionnaire was first piloted with a small group of senior students from the computing programs to refine wording and confirm face validity. Internal-consistency analysis on the full study sample yielded a Cronbach's α of 0.91, indicating excellent reliability and suggesting the 11 behavior items cohere well as a single measure of AI-supported programming practice.

The dataset was cleaned and verified, with all 50 respondents providing complete data for each behavior item. No missing values or response biases (such as straight-lining) were detected, as students varied their answers across behaviors. All items were positively phrased, so higher scores indicated more frequent use of recommended practices, and no reverse coding was necessary.

Operationalizing the Ideal Benchmark and Gap Computation

The ideal benchmark was the student response of *"Always"* (score of 5 on the 5-point scale) for each behavior, reflecting a strict pedagogical goal. For example, students should *always* test AI-generated code or plan before prompting. The gap for each behavior was calculated as follows: Gap = 5 – Student's Likert score, resulting in values from 0 (no gap) to 4 (maximum gap). For example, a student scoring *"Sometimes"* (3) for testing AI outputs would have a gap of 2.

This study also tested an alternative threshold (treating "Often" as acceptable), but the results mirrored the stricter benchmark. Using *"Always"* as the ideal better differentiates high performers. Gap scores were averaged across groups and analyzed using non-parametric methods (appropriate for ordinal data). This

study reports the percentage of students with gaps >0 (i.e., not answering *"Always"*), highlighting how widespread shortfalls are. This approach balances conceptual clarity with practical insights into AI usage habits.

Gap Analysis and Statistical Tests

The gap analysis is conducted in two steps. First, for RQ1 (overall shortfalls), we calculated the mean gap and percentage of students below the ideal for each behavior across all 50 students, allowing us to rank and visualize (Figure 1) which practices had the most significant shortfalls. Table 1 summarizes each behavior's mean gap and the proportion of students not meeting the ideal.

For RQ2 (complexity differences), we compared gap scores between High- and Low-Complexity groups using the Mann–Whitney U test, which is well-suited for small sample sizes and ordinal data. This test allowed us to assess whether the distribution of scores differed significantly between the two groups, even without assuming normality. While the mean gap scores were reported for each group and statistically significant differences were noted, the p-values were interpreted in context rather than applying strict corrections for multiple comparisons (American Statistical Association, 2016 ; Institute of Education Sciences, 2023; Sullivan & Fein, 2012). Because eleven behaviors were tested, applying a strict Bonferroni correction ($p < 0.05/11 = 0.0045$) to rule out the chance of finding at least one significant resuly by chance, would sharply raise the risk of Type II error, overlooking real effects (Perneger, 1998). Instead, the p-values were weighed alongside the magnitude and direction of each gap difference and the educational relevance of the behavior, so that statistically modest yet instructionally important trends were not dismissed.

This approach is especially appropriate in exploratory research, where the goal is to identify meaningful trends without overlooking potentially important results due to overly conservative statistical adjustments. This balanced approach pinpoints behaviours on which high-complexity students truly outperform (or underperform) their peers, supplying instructors with clear targets for scaffolding while avoiding the loss of meaningful signals that overly conservative corrections can cause.

All analyses were performed in Python (Pandas, SciPy), and the results are descriptive, given the self-report nature of the data. Gap values are interpreted as the distance from the ideal: a gap of 0 means *"Always"* performing the behavior, while higher gaps indicate less frequent practice. The next section, presents findings for all nine behaviors and discuss how task complexity relates to these gaps.

Results

Overall Shortfalls (RQ1)

Gap scores 5 minus the student's Likert rating, with 5 representing the ideal "Always"—reveal a strikingly uneven proficiency profile across the nine AI-support behaviors. Figure 1 shows the mean gap for each AI-support behavior. A gap of 0 means students on average already do the behavior Always; a gap of 4 means they Never do it. Behaviors are sorted from largest to smallest gap. Error bars show standard error of the mean gap.

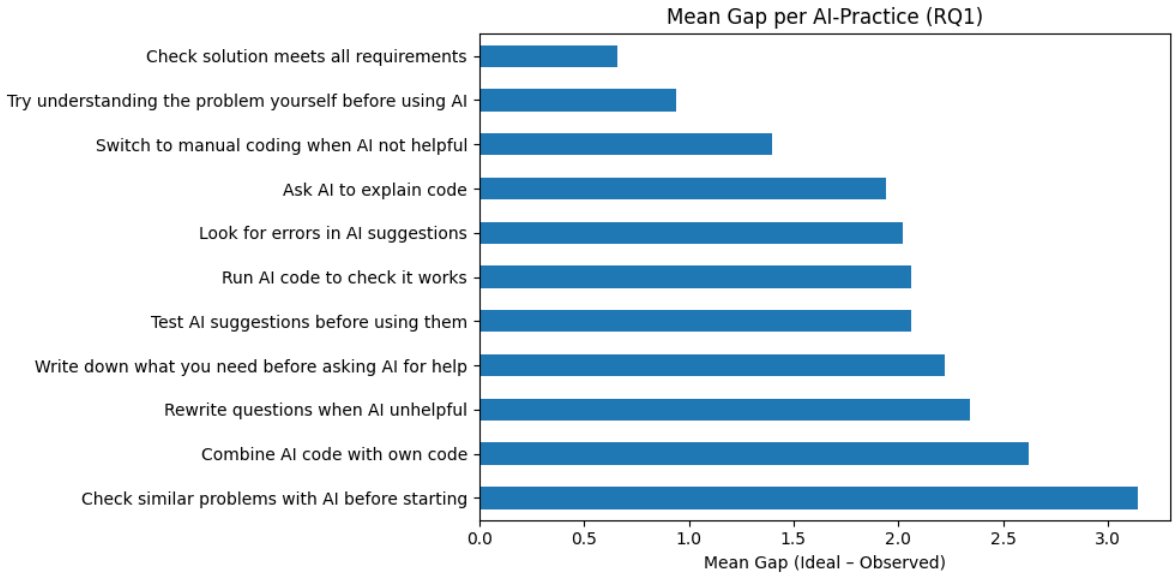

Figure 1. Mean shortfall (gap) for each AI-support behavior (N=50 students).

Table 1 shows the mean gap for each behavior and the percentage of students with a gap greater than or equal to 1 for each behavior. Appendix B shows the distribution of each gap score for each of the eleven behaviors.

Table 1 AI-Supported Programming Behavior Gap Survey Results

AI-Supported Behavior (Survey Item)	Mean Gap (0 = none, 4 = max)	% Students with Gap >=1	Mean Gap: Low-Complexity (n = 41)	Mean Gap: High-Complexity (n = 9)	Mann-Whitney (U, *p*)
Check similar problems with AI before starting	3.14	100 %	3.48	2.58	(446, 0.0011)
Combine AI code with own code	2.62	94 %	3.13	1.79	(491, 0.0001)
Rewrite question when AI answer is unhelpful	2.34	84 %	2.74	1.68	(418, 0.0112)
Write down needs before asking AI	2.22	74 %	2.45	1.84	(373, 0.108)
Run AI-generated code to check it works	2.06	72 %	2.58	1.21	(444, 0.0022)
Test AI suggestions before using	2.06	70 %	2.71	1.00	(476, 0.0002)
Look for errors in AI suggestions	2.02	76%	2.61	1.05	(469, 0.0004)
Ask AI to explain code you don't understand	1.94	72 %	2.68	0.74	(497,0.0001)
Switch to manual coding when AI isn't helpful	1.40	54 %	1.65	1.00	(365, 0.138)
Try to understand the problem yourself first	0.94	40 %	1.23	0.47	(339, 0.313)
Check solution meets all requirements	0.66	36%	0.65	0.68	(284, 0.812)

At the top of the deficit ladder sits *"Check whether the AI has already solved a similar problem"* (mean gap = 3.14). Every participant fell short of "*Always*" on this behavior, and fully two-thirds marked "*Rarely*" or "*Never*," indicating that students seldom treat AI as a research partner for hunting anlaogous solutions. This forethought lapse is pivotal: learners lose an efficient way to calibrate prompts or draw inspiration from solved exemplars without scouting for precedents.

The second-worst gap concerns "*Combine AI code with personal code*" (mean = 2.62, 94 % below ideal). Fewer than one in ten students reported "*Always*" integrating snippets, implying that most either copy AI output wholesale or ignore it rather than weaving it into their logic. Because modern development often involves stitching together multiple code sources, this weakness suggests that students are ill-prepared for authentic workflows in which the AI is a collaborator, not a one-shot answer engine. Close behind, "*Rewrite a question after an unhelpful answer*" registers a gap of 2.34. Seventy-per-cent of learners stop after a single inadequate response, forgoing the iterative prompting that experienced AI users rely on to refine output quality. A cluster of verification behaviors follows, each with mean gaps just above two points. Roughly three-quarters of the sample neglect running AI code (2.06), testing suggestions (2.06), and actively looking for errors (2.02). In concrete terms, only 24–30 % of students *always* execute these checks, leaving a large majority who trust AI output at least occasionally without systematic validation. Given well-documented hallucination and compilation errors in AI-generated code, this shortfall poses both learning and correctness risks.

Forethought planning is also weak: **"***Write down what you need before asking AI***"** posts a gap of 2.22. As shown in the gap distribution in APPENDIX B, over half of students(74 %) rarely commit their requirements to paper or pseudo-code before prompting. The absence of this seemingly simple step reflects a deeper metacognitive gap—students jump directly to asking without clarifying goals, increasing the likelihood of vague prompts and subpar answers. Encouragingly, the picture brightens for self-reliant actions. "*Switch to manual coding when the AI is not helpful"* records a smaller gap of 1.40; 44 % of learners *always* disengage from unhelpful output, and a further 21 % do so often. Even stronger is **"***Try to understand the problem yourself before using AI,***"** with a mean gap of just 0.94. Sixty percent of respondents claim they always attempt their understanding first, suggesting that foundational agency remains intact. The best-performed item overall—**"***Confirm that the final solution meets all requirements***"**—posts a minimal gap of 0.66, indicating that most students run a final mental or automated checklist before submission, even if earlier validation steps were partial.

Taken together, the data depict a group of students who retain core problem-solving instincts but struggle with AI-specific forethought, iteration, and verification. The magnitude of these shortfalls indicates the need for targeted instructional scaffolds: explicit training in exemplar scouting, prompt refinement, and systematic testing could convert sporadic, intuitive AI use into a disciplined, self-regulated workflow.

Task Complexity Differences (RQ 2)

To probe whether experience with demanding code influences AI-use quality, the sample was divided into a High-Complexity group—students who reported using AI "*often*" or "*always*" on programs exceeding 50 lines of code—and a Low-Complexity group, encompassing everyone else. Despite the High group's small size (n = 9), non-parametric tests reveal a striking, coherent advantage.

Forethought and planning behaviors improve first. High-complexity students cut the gap for "*rewriting an unhelpful prompt*" from 2.74 to 1.68 ($U = 418$, $p = .011$), showing they persist until the AI responds satisfactorily. Even more dramatic is "*combining AI code with personal code"*: the deficit shrinks from 3.13 to 1.79 ($U = 491$, $p < .0001$), suggesting that complex projects force learners to splice snippets rather than copy-paste wholesale. A parallel, though non-significant, trend appears for "*writing down needs before*

asking AI (2.45 → 1.84), hinting that the same students are also thinking ahead more deliberately. Verification habits strengthen next. High-complexity users almost halve their short-falls in *running AI code* (2.58 → 1.21) and *testing suggestions* (2.71 →1.00); similarly, their gap for *looking for errors* drops from 2.61 to 1.05 (all $p \leq .002$). These data imply that once tasks become intricate, students unquestioningly recognize the cost of trusting AI and adopt systematic checks—behaviors essential for professional robustness.

Conceptual engagement deepens as well. The most significant absolute gain appears in *asking the AI to explain unfamiliar code*, where the gap plummets from 2.68 to 0.74 ($U = 497$, $p = .0001$). Complex tasks prompt students to interrogate the AI for rationale rather than settle for a "black-box" answer, aligning with self-regulated learning's performance-monitoring phase. Where complexity fails to help is equally instructive. The two groups do not differ reliably in *switching to manual coding*, *understanding the problem first*, or *checking final requirements*. These actions were already moderate-to-strong across the cohort, suggesting a ceiling effect: students retain agency regardless of task size. However, one critical forethought behavior remains stubbornly weak: *checking whether the AI has solved a similar problem*. Even the High group posts a hefty gap of 2.58 (vs. 3.48 in the Low group, $p = 0.0011$), confirming that pre-searching is a universal blind spot.

Results on the effects of task complexity indicate that engaging AI on complex assignments acts as an informal tutor, sharpening iterative prompting, code integration, verification, and conceptual inquiry. However, engaging in complex problems does not teach learners to scout existing AI solutions, nor does it fully mitigate over-persistence with the tool. These residual gaps highlight domains where explicit instruction must supplement experiential learning.

Discussions

The results from the gap analysis sharpen the picture of how students self-regulate when programming with AI. The most severe shortfall remains forethought scouting: every respondent fell below the ideal of checking whether the AI has solved a similar problem (mean gap = 3.14). Skipping this step deprives learners of analogical examples that could guide both prompt quality and solution design; an omission squarely in SRL's planning phase (Zimmerman, 2002). Two further weaknesses dominate the performance phase. First, combining AI code with personal code posts a gap of 2.62; most students still treat AI output as all-or-nothing rather than weaving snippets into their logic. Second, verification is sporadic: running AI code, testing suggestions, and hunting for errors all hover around a gap of 2.0, indicating that roughly three-quarters of learners use AI output with only partial validation. These findings echo earlier work on novice over-trust in LLMs (Prather et al., 2024) and reveal that critical evaluation skills have yet to take root.

Encouragingly, two self-reliant behaviors show modest gaps. Switching to manual coding when AI falters (1.40) and trying to understand the problem first (0.94) suggest many students still recognize their agency—evidence of SRL's monitoring and control functioning at a baseline level. Checking final requirements registers the smallest gap (0.66), implying that end-stage review habits are comparatively solid.

Task Complexity as a Catalyst and a Caveat

Comparing high- and low-complexity users indicates the formative power of challenging work. Students who frequently applied AI to large programs (> 50 lines of code) cut the gap on several AI-support behaviors: integration dropped from 3.13 to 1.79 ($U = 491$, $p < .0001$); prompt-rewriting from 2.74 to 1.68 ($p = .011$); and all three verification items fell by roughly 1.4 points (all $p \leq .002$). Most striking, the gap for asking the AI to explain code plunged from 2.68 to 0.74, signaling a shift from black-box acceptance to

conceptual inquiry. These gains align with the "desirable difficulty" theory—more challenging tasks push learners into richer SRL planning cycles, monitoring, and adaptation (Bjork & Bjork, 2011; Weissgerber et.al. 2016).

However, complexity leaves two blind spots. First, even advanced users retain a large deficit on pre-searching (gap = 2.58), confirming that students across the board do not view AI as a research archive. Second, the groups do not differ reliably when switching to manual coding; the high group trends toward greater persistence with unhelpful AI (1.65 vs 1.00, p = .138). Thus, while complexity cultivates strategic depth, it can also breed dependence and does not automatically fix foundational planning lapses.

Instructional Implications

Educators should harness complex, authentic projects to reinforce iteration, integration, and verification but must overlay explicit scaffolds for the stubborn gaps in pre-search and adaptive disengagement. Table 1 summarises five low-overhead interventions mapped to the observed deficits. Together, they span all SRL phases: a pre-flight scan fortifies planning; split–merge tasks and prompt sprints enrich performance control; unit-test checkpoints institutionalize monitoring; and a three-prompt reflection rule strengthens adaptive self-evaluation. Implemented in concert, these measures can convert sporadic, tool-centered behavior into a balanced, self-regulated partnership with AI, leveraging the gains complexity confers while guarding against its liabilities.

Table 2. Targeted Scaffolds for Closing AI-Usage Gaps

Gap Addressed	Scaffold	Concrete Classroom Action
Forethought: no search for prior AI solutions	Pre-flight AI scan	Students submit a screenshot or 2-line summary of one AI-found exemplar before coding.
Integration: seldom merge AI & own code	Split–merge exercise	Instructor supplies an AI helper function; students must adapt and embed AI generated code in it, grading the merge diff/comments.
Prompt iteration: stop after one poor answer	Prompt-engineering sprint	15-minute in-class race to refine a weak prompt until output passes a rubric; bonus credit for best improvement.
Verification: partial or absent testing	Unit-test checkpoint	Every AI-assisted submission must include passing tests or run logs; small marks for evidence.
Over-reliance when AI fails	Three-prompt rule + reflection	After three failed AI attempts, students pivot to manual coding and write a 50-word reflection on the decision.

Limitations and Future Work

This study is limited by self-reported frequencies, learners may over- or under-estimate their habits, and by a modest, self-selected sample, particularly the high-complexity subgroup (n = 9). Demographic variables such as gender and age were not collected, which limits the ability to explore variation across subgroups, including traditional vs. non-traditional learners. Future studies should include these variables to examine potential moderating effects.

Despite these caveats, the project offers the first quantitative map of where students falter in AI-assisted coding and how difficulty modulates those gaps. It pinpoints four high-impact deficits, planning queries, prompt revision, verification, and code integration, giving instructors clear targets. The five low-overhead scaffolds proposed here translate directly into classroom practice, and the gap-score framework is ready for replication with richer data and broader populations.

The next steps move from diagnosis to remedy. Instructors can pilot AI-use workshops, split-merge labs, and prompting journals, then track whether gap scores shrink and whether smaller gaps predict higher code quality, grades, or concept gains. Qualitative interviews should probe why learners bypass certain behaviors, sharpening interventions. Finally, as tools evolve, the framework must evolve too; even "smarter" AI will still demand human planning, verification, and critical judgment, keeping AI literacy instruction essential.

Conclusion

This study provided a quantitative look at how students currently engage (or fail to engage) in recommended practices when programming with AI assistance. The "gaps" we identified, particularly in upfront planning and iterative refinement, highlight that students are often *not* using AI tools as effectively as they could. The encouraging news is that these are addressable gaps: With proper guidance, curriculum design, and experience, students can learn to use AI in a more self-regulated, productive manner. By "minding the gap" and proactively teaching AI-support behaviors, educators can help students solve problems with AI and learn more from the process. As Zimmerman (2002) noted, self-regulation is teachable and is a key differentiator in successful learners. In the era of AI, self-regulation takes on new forms; it means knowing how to collaborate with a non-human partner in one's learning. Our findings offer concrete targets for teaching interventions to foster that capability.

Bridging the gap in AI-supported programming practices will better prepare students for a future where human-AI collaboration is the norm. This work lays a foundation for improving AI literacy in computing education by quantifying these shortfalls and their predictors. We envision a classroom where using an AI assistant is second nature to students and where they consistently apply the best practices to use it wisely. When such best practices become widespread, the gap will truly be narrowed, and the promise of AI in education – augmenting human learning, not replacing it.

References

Acosta-Enriquez, B. G., Arbulú Ballesteros, M. A., Huamani Jordan, O., Lopez Roca, C., & Saavedra Tirado, K. (2024). Analysis of college students' attitudes toward the use of ChatGPT in their academic activities: effect of intent to use, verification of information and responsible use. BMC Psychology, 12. https://doi.org/10.1186/s40359-024-01764-z

Al Haque, E., Brown, C., LaToza, T. D., & Johnson, B. (2024). The evolution of information seeking in software development: Understanding the role and impact of AI assistants. arXiv. https://doi.org/10.48550/arXiv.2408.04032

Alrayes, A., & Aljohani, N. R. (2024). Student-AI Interaction: A Case Study of CS1 students. arXiv. https://arxiv.org/abs/2407.00305

American Statistical Association. (2016). ASA statement on statistical significance and p-values. Retrieved 05/16/2025. https://www.amstat.org/asa/files/pdfs/p-valuestatement.pdf

Bjork, E. L., & Bjork, R. A. (2011). Creating desirable difficulties to enhance learning. In M. A. Gernsbacher, R. W. Pew, L. M. Hough, & J. R. Pomerantz (Eds.), Psychology and the real world: Essays illustrating fundamental contributions to society (pp. 56–64). Worth Publishers.

Bosch, N., D'Mello, S. (2017).The Affective Experience of Novice Computer Programmers. Int J Artif Intell Educ 27, 181–206. https://doi.org/10.1007/s40593-015-0069-5

Clark, R. E., & Estes, F. (2008). Turning research into results: A guide to selecting the right performance solutions. Information Age Publishing.

Darvishi, A., Khosravi, H., Sadiq, S., Gasevic, D., & Siemens, G. (2024). Impact of AI assistance on student agency. Computers & Education, 210, 104967. https://doi.org/10.1016/j.compedu.2023.104967

Fenu, G., Galici, R., Marras, M., & Reforgiato, D. (2024). Exploring student interactions with AI in programming training. In Adjunct Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization (UMAP Adjunct '24) (pp. 555–560). Association for Computing Machinery. https://doi.org/10.1145/3631700.3665227

Institute of Education Sciences. (2023, March 28). Statistically significant doesn't mean meaningful. Retrieved 05/16/2025. U.S. Department of Education. https://ies.ed.gov/learn/blog/statistically-significant-doesnt-mean-meaningful

Kloub, L., & Gupta, A. (2024). ChatGPT in computer science education: Exploring benefits, challenges, and ethical considerations. Proceedings of the 2024 ASEE Northeast Section Conference. https:// https://doi.org/10.18260/1-2--45758

Knoth, J., Chiu, M. M., & Mason, S. (2024). DeBiasMe: De-biasing human-AI interactions with metacognitive AIED interventions. In Workshop: AI Augmented Reasoning, CHI 2025. arXiv preprint. https://arxiv.org/abs/2504.16770

Lan, M., Zhou, X.(2025) A qualitative systematic review on AI empowered self-regulated learning in higher education. npj Sci. Learn. 10, 21 (2025). https://doi.org/10.1038/s41539-025-00319-0

Laupichler, M. C., Aster, A., Schirch, J., & Raupach, T. (2022). Artificial intelligence literacy in higher education: A systematic review. Computers & Education, 189, 104596. https://doi.org/10.1016/j.caeai.2022.100101

Lo, C. K. (2023). What is the impact of ChatGPT on education? A rapid review of the literature. Education Sciences, 13(4), 410. https://doi.org/10.3390/educsci13040410

Luckin, R., Cukurova, M., Kent, C., & du Boulay, B. (2022). Empowering educators to be AI-ready. Computers and Education: Artificial Intelligence, 3, Article 100076. https://doi.org/10.1016/j.caeai.2022.100076

Microsoft. (2024). AI prompting guidelines for educators. Retrieved 05/16/2025 from https://ecampusontario.pressbooks.pub/conestogagenaiguidebook/chapter/1-3-prompting/

Mozannar, H., Bansal, G., Fourney, A., & Horvitz, E. (2024). Reading between the lines: Modeling user behavior and costs in AI-assisted programming. In CHI '24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (Article No. 142, pp. 1–16). Association for Computing Machinery. https://doi.org/10.1145/3613904.3641936

Naamati-Schneider, L., & Alt, D. (2023). Beyond digital literacy: The era of AI-powered assistants and their impact on education. Education and Information Technologies, 28(6), 6895–6908. https://doi.org/10.1007/s10639-024-12694-z

Newman, R. S. (1994). Adaptive help seeking: A strategy of self-regulated learning. In D. H. Schunk & B. J. Zimmerman (Eds.), Self-regulation of learning and performance: Issues and educational applications (pp. 283–301). Lawrence Erlbaum Associates.

Perneger T. V. (1998). What's wrong with Bonferroni adjustments. BMJ (Clinical research ed.), 316(7139), 1236–1238. https://doi.org/10.1136/bmj.316.7139.1236.

Prather, J., Denny, P., Leinonen, J., Becker, B. A., Albluwi, I., & Reeves, B. N. (2024). The Widening Gap: The benefits and harms of generative AI for novice programmers. Proceedings of the ACM Conference on International Computing Education Research (ICER). https://doi.org/10.1145/3632620.3671116

Prasad, P., & Sane, A. (2024). A self-regulated learning framework using generative AI and its application in CS educational intervention design. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024) (pp. 1–7). ACM. https://dl.acm.org/doi/10.1145/3626252.3630828

Rahman, M. M., & Watanobe, Y. (2023). ChatGPT for Education and Research: Opportunities, Threats, and Strategies. Applied Sciences, 13(9), 5783. https://doi.org/10.3390/app13095783

Roll, I., & Wylie, R. (2016). Evolution and revolution in artificial intelligence in education. International Journal of Artificial Intelligence in Education, 26(2), 582–599. https://doi.org/10.1007/s40593-016-0110-3

Sun, D., Boudouaia, A., Zhu, C. et al. (2024). Would ChatGPT-facilitated programming mode impact college students' programming behaviors, performances, and perceptions? An empirical study. Int J Educ Technol High Educ 21, 14 (2024). https://doi.org/10.1186/s41239-024-00446-5

Sullivan, G. M., & Feinn, R. (2012). Using Effect Size-or Why the P Value Is Not Enough. Journal of graduate medical education, 4(3), 279–282. https://doi.org/10.4300/JGME-D-12-00156.1

Tang, C., Ng, V., Leung, H., & Yuen, J. (2024). AI-generated programming solutions: Impacts on academic integrity and good practices. Proceedings of the 16th International Conference on Computer Supported Education (CSEDU 2024), 478–485. https://doi.org/10.5220/0012563600003693

Weissgerber, S. C., Reinhard, M.-A., & Schindler, S. (2016). Study harder? The relationship of achievement goals to attitudes and self-reported use of desirable difficulties in self-regulated learning. Journal of Psychological and Educational Research, 24(1), 42–60.

Yilmaz, R., & Yilmaz, K. F. G. (2023). The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. Computers and Education: Artificial Intelligence, 4. https://doi.org/10.1016/j.caeai.2023.100147

Zhou, C. (2023). Integration of modern technologies in higher education on the example of artificial intelligence use. Educ Inf Technol 28, 3893–3910 (2023). https://doi.org/10.1007/s10639-022-11309-9

Zimmerman, B. J. (2002). Becoming a self-regulated learner: An overview. Theory Into Practice, 41(2), 64–70.

Appendix A

Table A – Programming with AI tools Survey

1.Your programming experience in years.					
2. AI tool experience in years					
3. Current academic standing					
4. Major					
Questions- Usage Behavior: **In your recent programming assignments, how often did you:**	Never **[1]**	Rarely (1-2 times per assignment/ project) **[2]**	Sometimes (3-4 times per assignment/ project) **[3]**	Often (5-6 times per assignment /project) **[4]**	Always (7+ times per assignment / project) **[5]**
5a.[Check if similar problems can be solved using AI before starting?]					
5b. [Write down what you need before asking AI for help?]					
5c. [Try understanding the problem yourself before using AI?]					
5d. [Test AI's suggestions before using them in your code?]					
6a. [Rewrite your questions/prompts when AI gives unhelpful answers?]					
6b. [Run AI's code to check if it works?]					
6c. [Combine pieces of AI code with your own code?]					
6d. [Ask AI to explain code you don't understand?]					
7a. [Look for errors in AI's solution?]					
7b. [Switch to coding yourself when AI isn't helpful?]					
7c. [Check solution meets all the requirements ?]					
Questions- Complexity: I use Al tools for:					
7a. [Simple programming tasks < 10 lines of code)]					
7b. [Medium complexity Tasks (10-50 lines of code)]					
7c. [Complex Programming Tasks (> 50 lines of code)]					

Issues in Information Systems
Volume 26, Issue 1, pp. 458-471, 2025

Appendix B

Table B – Frequency distribution of gaps for each AI-support behavior.