2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
Paper ID #51183

The Redesign of an ECE Programming Course: The Integration of Active Learning, Artificial Intelligence, Immediate Feedback, and Unlimited Submissions

Dr. Rajeev Sahay, University of California, San Diego

Dr. Rajeev Sahay received the B.S. degree in electrical engineering from The University of Utah, Salt Lake City, UT, USA, in 2018, and the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2021 and 2022, respectively. Currently, he is a faculty member in the Department of Electrical and Computer Engineering at the University of California San Diego (UCSD). Dr. Sahay was the recipient of the Purdue Engineering Dean's Teaching Fellowship as well as the best teaching award in the Department of Electrical and Computer Engineering at UCSD. Dr. Sahay's teaching interests include data science, machine learning, analog circuits, computer architecture, and programming languages in C, C++, and Python. His research interests lie in the intersection of networking and machine learning, especially in their applications to wireless communications and engineering education.

The Redesign of an ECE Programming Course: The Integration of Active Learning, Artificial Intelligence, Immediate Feedback, and Unlimited Submissions

Abstract

This paper presents the redesign and iterative refinement of an undergraduate ECE course on object-oriented programming and data structures in C++. Across several academic quarters, the course incorporated four major interventions informed by student feedback and evolving trends in software engineering education. These included adding participation-based clicker questions, restructuring programming assignments to allow automated feedback with unlimited resubmissions, introducing collaborative Discussion sessions with immediate feedback, and integrating the use of large language models (LLMs). Together, these components created an environment where students could apply and master challenging concepts through low-stakes assignments in a supportive and flexible learning environment while preserving rigor through high-stakes closed-internet, independent exams. Survey data indicated that students valued the interactivity, appreciated opportunities to revise work until mastery, and generally reported low reliance on AI for completing assignments. Overall, the redesign demonstrates how thoughtful, formative practices can modernize core programming courses and better align them with student needs and LLM integration.

1 Introduction

Undergraduate electrical and computer engineering (ECE) curricula increasingly rely on students' abilities to write robust, modular, and scalable software (e.g., in fields such as embedded systems, digital signal processing, and advanced software engineering) [1–3]. As modern engineering practice integrates embedded systems, intelligent devices, and data-driven decision-making, students must be able to reason about abstraction, manage program complexity, and work fluently with data structures. At many institutions, the second programming course, typically focused on object-oriented programming (OOP) and foundational data structures, serves as a critical bridge between introductory programming and advanced coursework in algorithms, software design, embedded systems, and controls. Ensuring that students achieve genuine competence at this stage is essential not only for later academic success but also for alignment with industry expectations around software engineering practices for electrical engineering and computer engineering

graduates [4–6].

However, teaching this material effectively presents several challenges. Students often encounter steep conceptual jumps when moving from basic procedural programming to object-oriented design, memory management, and pointer-based data structures. Traditional lecture-based formats, in which students passively watch code demonstrations and complete weekly programming assignments, can inadvertently obscure these conceptual transitions [7]. Moreover, introductory programming courses often struggle with broad variability in student preparedness, high cognitive load, and limited opportunities for timely feedback. At the same time, broader shifts in engineering education, such as the increasing student usage of large language models (LLMs) and artificial intelligence (AI) tools in general, have pushed instructors to reconsider how best to foster mastery while maintaining academic integrity and independent assessment [8–12].

In response to these challenges and the evolving landscape of engineering education, we redesigned our second programming course in C++ in the Department of Electrical and Computer Engineering at the University of California San Diego (UCSD) through a series of iterative, multi-quarter interventions. The redesign introduced four major structural changes. First, lectures were transformed through the integration of clicker questions as low-stakes formative assessments, emphasizing active learning and immediate conceptual checks [13]. Second, programming assignments were restructured to provide automated feedback with unlimited resubmissions prior to a deadline, supporting mastery learning while reducing student frustration [14]. Third, weekly Discussion (supplemental instruction) assignments were redesigned to encourage collaboration under teaching assistant (TA) guidance, blending cooperative problem-solving with the benefits of automated feedback [14, 15]. Finally, the course incorporated the responsible use of AI, acknowledging their increasing role in engineering education, and specifically in programming, while providing guardrails to prevent overdependence from students [16].

This combination of interventions distinguishes the course redesign from typical incremental updates often applied to programming classes for non-computer science majors. Rather than modifying a single component (e.g., adding clickers or adopting auto-graders), the redesign simultaneously restructured lecture pedagogy, programming assignments, collaborative learning environments, and expectations around AI usage. These changes were grounded in evidence-based instructional practices and were iteratively refined through multiple quarters of student feedback and instructor reflection. The result is a student-centered course that is highly effective for learning under a unified instructional model that balances flexibility with accountability: students receive extensive formative support and multiple pathways to mastering challenging concepts, but summative assessments emphasize independent problem-solving without technology (AI, internet, or computers in general) aids.

This paper presents the design, implementation, and evaluation of this multi-component redesign of an OOP and data structures course for ECE majors. By documenting the motivation, instructional choices, and student responses across several quarters, we aim to contribute to ongoing discussions about how engineering instructors can modernize core courses in ways that are both pedagogically effective and aligned with evolving engineering education practice. The insights offered here may support educators seeking to integrate formative assessment, automated

feedback, collaborative learning, and responsible AI use into their own course ecosystems.

2 Methods

This section describes the structure of the course, the instructional interventions introduced during the redesign, and the multi-quarter implementation of those interventions. We begin by outlining the content covered in the course and the expected learning outcomes for students. The subsections that follow describe the four major components of the redesign: the integration of clicker-based formative assessments, the restructuring of programming assignments around automated grading and mastery learning, the introduction of collaborative Discussion assignments, and the incorporation of responsible AI use.

2.1 Course Topics

The redesigned course is positioned as the second programming course in the ECE curriculum. Students entering the course are expected to have prior experience with basic procedural programming, including variables, control structures, functions, arrays, memory management, and simple problem-solving strategies (taught in C). The course content is structured to review and reinforce these foundations while introducing object-oriented programming, standard libraries, algorithms, and data structures. In addition, the course aims to cultivate engineering-relevant computational reasoning, including modular design, efficiency considerations, and debugging strategies. These objectives situate the course as an essential prerequisite for advanced coursework such as software engineering and embedded systems courses.

Table 1 outlines the week-by-week lecture schedule throughout the ten week course, where two lecture sessions were held each week. The pacing of the course was intentionally designed to balance prerequisite concepts with advanced topics. Early weeks focus on reviewing foundational programming skills, ensuring that students with inadequate preparation (or students that took the prerequisite course much earlier) were ready to apply procedural programming concepts to more advanced topics. During this time, the course focused on material that students consistently struggle with in prerequisite courses such as pointers and dynamic memory allocation. Subsequent weeks introduce increasingly sophisticated material, including objects and classes, the standard template library, linked lists, stacks, queues, recursion, binary search trees, exception handling, and time complexity.

2.2 Redesign of Lecture Content

The first major intervention involved redesigning lectures to incorporate examples demonstrating the applications of theoretical concepts and clicker questions as low-stakes formative assessments. Both changes were motivated by attempting to make lectures be more engaging and active for students compared to traditional lecture formats, which often feel passive and difficult to follow, particularly when complex programming concepts are introduced. This active learning

Week	Lecture 1	Lecture 2
1	Data Types and I/O	Control Structures
2	Strings & Functions	Pointers and Dynamic Memory Allocation
3	Structs and Vectors	Classes I
4	Classes II	Encapsulation and Inheritance
5	Polymorphism and Abstraction	Templates
6	Review	Exam
7	Standard Template Library I	Standard Template Library II
8	Linked Lists	Stacks and Queues
9	Recursion	Binary Search Trees
10	Time Complexity (searching and sorting)	Exception Handling

Table 1: The lecture schedule and topics of the course.

activity via clicker questions provided two benefits: (i) they provided a mechanism for students to actively engage with material in real time and (ii) they gave instructors visibility into student understanding. The questions were intentionally designed to focus on conceptual reasoning rather than syntactic details. For example, most questions would present code snippet and ask for the output, ask about the value of a variable at the end of the program, or ask about the output of an algorithm when applying it to a data structure. Fig. 1 shows a sample clicker question. This ensured broad student participation as opposed to live programming challenges, which were implemented in one of the quarters of the iterative redesign process an ultimately consumed a large amount of lecture time without effective learning or student engagement. Grading the questions solely for participation further reduced anxiety and emphasized learning over performance in a low-stakes setting.

The clicker questions were embedded throughout the lecture rather than concentrated at the beginning or end. This allowed them to function as checkpoints that segmented lectures into cognitively manageable pieces. After students submitted responses, during which they were allowed to discuss their approach with their peers, the instructor reviewed the solution, allocating additional time for questions with lower rates of correct answers. This structure supported peer instruction by providing natural opportunities for students to articulate their reasoning, debate alternatives, and collaboratively refine their understanding. The approach also allowed the instructor to decide, in real-time, whether a concept required additional explanation before moving on.

Each question was crafted to reveal specific misunderstandings that could then be addressed immediately. The iterative redesign process enabled the instructor to refine questions across quarters based on observed student responses and misconceptions over longer periods of time. This iteration ensured that the clicker questions remained relevant and addressed student concerns before they were raised.

The logistical policies associated with clicker use were intentionally kept simple to encourage participation. Students were required to answer a minimum number of questions per class, though the exact threshold were adjusted across sessions (due to the varying number of questions per lecture). Because the questions were graded for participation rather than correctness, students

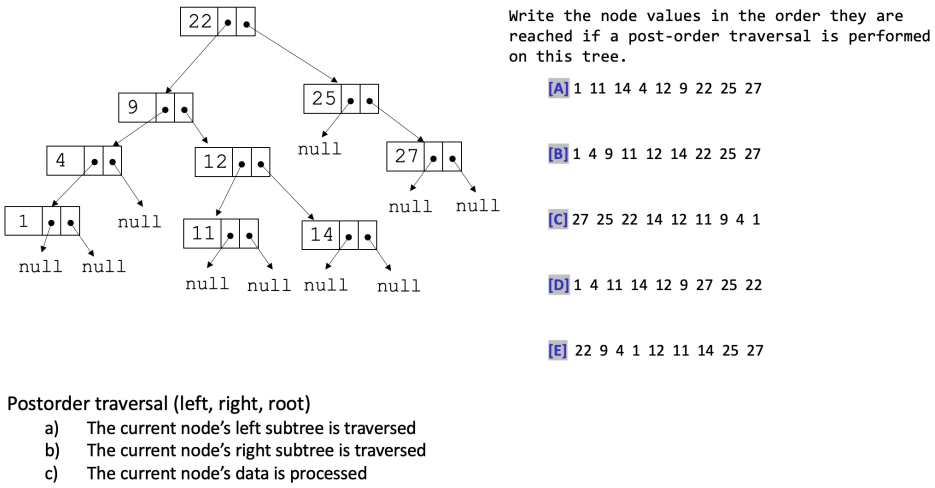

Figure 1: An example clicker question testing conceptual understanding in the Binary Search Tree lecture.

were encouraged to attempt answers even when unsure, reinforcing the low-stakes nature of the intervention. This emphasis on participation contributed to a classroom culture where mistakes were treated as natural components of the learning process.

2.3 Redesign of Programming Assignments

The second major intervention centered on restructuring programming assignments to provide automated grading, immediate feedback, and unlimited resubmissions. Prior versions of the course relied on weekly assignments submitted once for manual grading, which often resulted in delayed feedback and limited opportunities for students to correct misunderstandings as students rarely revisited their assignments after receiving their scores. The new design adopted an auto-grading infrastructure using Ed Stem, where the instructional team designed specific test cases with custom error messages for each assignment. Students could submit their programs for immediate feedback after finishing portions of it (e.g., different functions or classes), allowing students to iteratively refine their code. This shift aligned with mastery-learning principles, prioritizing student growth over one-time performance.

The policies governing the programming assignments emphasized independence. Students were required to complete these assignments individually, and the course established clear guidelines for what forms of collaboration or assistance were permitted. Specifically, students were allowed to use the internet as well as LLMs, but they were not allowed to paste code into their editors or work with other people (other than members of the instructional staff). To support independent work, the instructor and teaching assistants held office hours to help with debugging and additional errors. This policy ensured that assignments functioned as authentic demonstrations of student capability. The majority of students enjoyed this model and preferred learning from members of the teaching staff over LLMs (further discussed in Sec. 2.5 and Sec. 3).

2.4 Redesign of Supplemental Instruction

The third intervention involved redesigning weekly supplemental instruction (i.e., Discussion sessions) to include assignments, which reviewed the material covered in the previous week. These Discussion assignments provided automated feedback and allowed unlimited submissions, similar to the programming assignments. However, unlike programming assignments, which students were required to complete independently, students were allowed and encouraged to work together on Discussion assignments under the supervision of a teaching assistant (TA). This allowed students to learn in the way best fit for them (i.e., with a partner, with a TA, independently, or a combination of these). The Discussion sessions emphasized both conceptual understanding (via a TA-led review during the first half of the Discussion period) and applications of theoretical concepts (via the required Discussion assignment during the second half of the Discussion period).

2.5 Integration of Artificial Intelligence

The fourth intervention involved integrating the responsible use of AI tools, particularly LLMs, into the course. This decision reflected the rapid adoption of AI-assisted programming tools in engineering education [9]. At the same time, the instructional team recognized the potential risks associated with over-reliance on AI, including diminished problem-solving skills and reduced understanding of core concepts. Moreover, students cannot be expected to decipher trustworthy and accurate content on topics they are learning for the first time. As a result, the course adopted a balanced approach that encouraged thoughtful use of AI while establishing clear boundaries to promote learning and prevent misunderstandings from incorrectly generated content. This intervention represented a substantial departure from traditional programming courses, which often ignore or prohibit AI tools without explicit guidance. Note that students were not given an LLM trained on specific course material. Instead, consistent with prior work, they were permitted to use open-source LLMs such as ChatGPT [16].

Students were provided with guidelines outlining appropriate and inappropriate uses of AI during the course. Specifically, students were encouraged to get occasional help from AI but ultimately apply solutions by themselves (since pasting code from anywhere including AI into assignments was prohibited). These guidelines emphasized that AI could serve as a resource for clarifying concepts or generating starter ideas, but should not be used to produce complete solutions for assignments. The instructional team stressed the importance of verifying and understanding any AI-generated output, aligning with the broader educational goal of fostering independent reasoning. To this end, the process of code co-generation with AI was never explicitly taught during lecture. All lectures consisted of instructor-made content and problem-solving from scratch without the use of AI.

The policy for AI use varied across the different components of the course. For example, AI assistance was permitted in certain contexts (e.g., programming assignments and Discussion assignments) but prohibited in others (e.g., exams). The rationale for these distinctions was to allow the instructor to fairly assess independent learning. Namely, students were expected to be

Category	Weight
In-class participation (clickers)	5%
Discussion	5%
Programming Assignments	20%
Midterm Exam	35%
Final Exam	35%

Table 2: The weight of each course component on the final grade.

able to independently implement and reason about code, as they were asked to do on exams. The assignments were provided as a tool to help student learning. The use of AI was permitted as a tool to help learning, but ultimately, mastery was measured by performance on independent assessments without the use of AI.

2.6 Student Assessment

Student learning in the redesigned course was evaluated through an assessment structure that emphasized fairness, rigor, and independent mastery of core programming concepts. To ensure that grades reflected each student's own understanding, the course relied heavily on traditional paper-and-pencil exams. However, since the focus of the course and the exams were on reasoning rather than regurgitation, the exams were open-book and open-note. Students were allowed to reference any paper materials during the exams, but they could not access any electronics or the internet (including LLMs). The exams served as the primary mechanism for evaluating students' ability to reason about object-oriented design, analyze data structures, and solve algorithmic problems without technological support. Over the ten-week course, two non-cumulative exams were administered.

To complement these high-stakes assessments, the course incorporated several low-stakes formative assessments, consisting of programming assignments, participation based in-class clicker questions, and Discussion assignments. These assessments were designed to support learning rather than drive grades, allowing students to explore concepts, receive feedback, and build skill over time. To encourage completion and participation in these low-stakes assessments, all low-stakes assessments were factored into the final course grade.

This grading structure ensured that while students were given ample opportunities for practice and feedback, their final grades ultimately reflected demonstrable individual mastery. By placing the greatest weight on closed-environment exams and limiting the influence of AI-vulnerable assignments, the assessment design aligned closely with the goals of the course redesign: promoting accountability, supporting active learning, and creating a fair and transparent system for evaluating student learning. The final distribution of grading categories is summarized in Table 2. Numerical grades were mapped to letter grades using the standard scale (e.g., $> 93\%$ is an A, $> 90\%$ is an A-, etc.). The rationale for this mapping scheme was based on the grade weights; students with very high or perfect scores on low-stakes formative assessments, making up 30% of the course grade, along with low scores on independent exams, making up 70% of the course grade, demonstrated a low ability to independently reason about the course content and

Intervention	S24	F24	S25	SS25	F25
Redesign of Programming Assignments	X	✓	✓	✓	✓
Redesign of Supplemental Instruction	X	✓	✓	✓	✓
Clicker Questions	✓	✓	X	✓	✓
Integrations of Artificial Intelligence	X	✓	✓	✓	✓

Table 3: A summary of the various interventions and which quarters they were withheld in.

would be at high risk of failing the course.

3 Results and Discussion

This section presents the outcomes of the course redesign based on data collected across five academic quarters: Spring 2024 (S24), Fall 2024 (F24), Spring 2025 (S25), Summer Session 2025 (SS25), and Fall 2025 (F25). Course surveys were administered at the conclusion of each of the five quarters of the course. To isolate the impact of each intervention, different interventions were withheld from various quarters, with most recent quarters jointly using each intervention. Table 3 details the interventions that were used and withheld during each quarter of the study. Note that clicker questions were removed from the S25 offering to assess its impact on lecture engagement. Student evaluations were used to assess the impact of the course redesign. Table 4 summarizes the enrollment and student response rate to these surveys in each quarter. In these course-specific student evaluations, students were asked about the effectiveness of each of the four interventions in relation to their learning. Course surveys were administered over five quarters of the course. Table 4 summarizes the enrollment and student response rate in each quarter.

The following select questions are analyzed in this study as they relate to each of the four interventions introduced in the course redesign. Their corresponding possible answers (as worded to students) are shown with each question.

- *Q1*: How would you rate your overall course satisfaction with ECE 17? Possible answers: 5 (very satisfied), 4 (satisfied), 3 (neutral), 2 (unsatisfied), 1 (very unsatisfied).
- *Q2*: How long did you take to complete each PA on average? Possible answers: 5 – 0-2 hours, 4 – 2-4 hours, 3 – 4-6 hours, 2 – 6-8 hours, 1 – 8+ hours.
- *Q3*: The difficulty of the Programming Assignments (PAs) are appropriate. Possible answers: 5 Strongly agree (I usually complete the PAs without too much struggle and I feel like I learn from them), 4 Agree (I sometimes need help with the PAs but I still learn from them), 3 Neutral (the PAs are too easy OR I feel like I don't learn from them), 2 Disagree (the PAs are challenging and hard for me to learn from), 1 Strongly disagree (the PAs are extremely challenging and I never learn from them).
- *Q4*: I feel like the lectures are useful and I learn from them. Possible answers: Strongly agree (5), Agree (4), Neutral (3), Disagree (2), Strongly disagree (1).
- *Q5*: The Discussion assignments helped me better understand the course content. Possible answers: Strongly agree (5), Agree (4), Neutral (3), Disagree (2), Strongly disagree (1).

Quarter	S24	F24	S25	SS25	F25
Enrollment	64	57	76	44	54
Response	50	50	57	40	46
Response Rate	78.13%	87.71%	75%	90.90%	85.19%

Table 4: The enrollment and response rate over each quarter of the iterative redesign process.

Question	S24	F24	S25	SS25	F25
Q1	4.56 ± 0.64	4.68 ± 0.51	4.47 ± 0.60	4.73 ± 0.45	4.89 ± 0.31
Q2	2.96 ± 0.81	3.76 ± 0.72	3.28 ± 0.96	3.35 ± 1.05	3.37 ± 0.97
Q3	4.20 ± 0.67	4.30 ± 0.54	4.09 ± 0.71	4.20 ± 0.56	4.28 ± 0.66
Q4	4.56 ± 0.79	4.50 ± 0.74	4.51 ± 0.54	4.78 ± 0.48	4.76 ± 0.57
Q5	3.68 ± 1.28	4.38 ± 0.85	4.19 ± 0.74	4.20 ± 0.85	4.35 ± 0.77
Q6	N/A	2.74 ± 1.27	2.86 ± 1.11	3.00 ± 0.78	2.45 ± 0.96
Q7	N/A	3.28 ± 1.34	3.51 ± 1.10	3.75 ± 0.95	3.09 ± 1.33

Table 5: The average response (with its associated standard deviation) of each question presented to students at the end of each quarter along with the average final percentages in each course.

- *Q6*[1]: I frequently used AI (e.g., ChatGPT, copilot, etc.) to help me with the Programming Assignments (PAs). Possible answers: 5 Strongly agree (I used AI for help on every PA), 4 Agree (I used AI for help on most PA), 3 Neutral (I seldom used AI for help but did not use it regularly or rely on it), 2 Disagree (I used AI occasionally, e.g., on a couple PAs, but not regularly), 1 Strongly Disagree (I never used AI for help with the PAs.
- *Q7*[2]: I frequently used AI (e.g., ChatGPT, copilot, etc.) to help me understand course concepts (i.e., used it to get help with conceptual material but not necessarily programming help). Possible answers: 5 Strongly agree (I used AI for help understanding several concepts), 4 Agree (I used AI for help understanding several concepts), 3 Neutral (I seldom used AI for help but did not rely on it), 2 Disagree (I used AI occasionally, e.g., to understand a couple concepts, but not regularly), 1 Strongly Disagree (I never used AI for help understanding the material and relied on the instructor and/or TAs).

Table 5 summarizes the student response results. In general, student learning satisfaction (*Q1*) was highest with the lowest variance in F24, SS25, and F25 when all interventions were jointly used, indicating the efficacy of the combination of interventions applied simultaneously. Furthermore, analyzing the mean response to *Q1* between S24 and F25 (before and after refinement of the iterative redesign) using an unpaired t-test yields a p-value of 0.0021, indicating that *the difference in mean in response to student satisfaction before and after the adoption and refinement of all applied interventions is statistically significant.* When one or more of these interventions was removed (e.g,. in S24 and S25), student satisfaction in the course was directly impacted as shown in Table 5.

The redesigned programming assignments contributed to more efficient student learning as shown

[1]This question was only asked in quarters in which AI was permitted for use according to Table 3.

[2]This question was only asked in quarters in which AI was permitted for use according to Table 3.

in response to *Q2* in Table 5. Specifically, in S24 prior to the redesign of the programming assignments, students spent more time on average on each assignment whereas, as shown in response to *Q2* for all subsequent quarters after S24, students spent on average less time on each assignment. It is important to note here that the assignments themselves did not change as a result of the intervention. Only the format in which they were presented to students changed in that, after redesigning the assignments, they were given to students in a format that allowed for unlimited resubmissions and immediate feedback. Since the assignments themselves did not change, this shows that students were able to successfully learn the content in slightly less time (as also indicated by average assignment scores being close to 100 due to the multiple attempts). This also explains the consistent response to the difficulty of the assignments as shown in response to *Q3* in Table 5, where the average difficulty and variance remained similar across each quarter, further corroborated by statistically insignificant results across quarters in differences in means (as expected since the assignments themselves were unchanged).

Overall, students found lecture sessions consistently constructive to their learning over multiple quarters as shown in response to *Q4* in Table 5, where students' responses were consistently high each quarter. Thus, to assess the impact of the inclusion and exclusion of clicker questions, qualitative survey responses were analyzed. In this capacity, the quarter in which the clickers were removed (S25), adding in-class questions was the most requested change asked from students. Similarly, in quarters in which clickers were used (S24, F24, SS25, and F25), this was noted by more than half of the responses as one of the most helpful learning tools during lecture. In addition, the instructor noted that in-class engagement was higher, which was observed via a higher attendance rate and more time during class dedicated to in-class questions from students during quarters with clicker questions (despite approximately consistent enrollment across quarters). Finally, note that the quarter in which the clicker questions were removed, the questions from the clickers were still presented as in-class examples that the instructor would work through as part of the lecture.

The redesign of the Discussion assignments had a noticeable impact on student learning as shown in response to *Q5* in Table 5. Specifically, in S24 prior to the redesign of the Discussion assignments, the average response to *Q5* was 3.68 with a large variance, indicating that Discussion assignments and sessions did not directly help students with reinforcing, reviewing, or learning the course content. However, the average response to same question in future quarters, which all introduced redesigned Discussion assignments, was 4.19 or greater with a smaller variance in comparison so S24, showing that more students regularly found these redesigned assignments helpful for learning. Moreover, the statistical comparison between the mean in response to *Q5* between S24 and F25 (before and after refining the redesigned the Discussion assignments) using an unpaired t-test yields a p-value of 0.0021, indicating that *the redesigned Discussion assignments had a statistically significant impact on student learning*.

Regarding AI (*Q6* and *Q7*), the results show that students remain approximately neutral with their reliance on AI throughout the course. More specifically, student reliance on AI specifically for programming assignments (*Q6*), remains less widespread in comparison to student AI usage for general conceptual support (*Q7*). The overall lower utilization and reliance on AI reflects the course's position on AI; students are permitted to use AI to help deepen their conceptual understanding, but efficient AI usage (e.g., prompt engineering) is not taught during lecture and,

hence, students are expected to learn the content independently. As a result, students appear to avoid relying on AI tools throughout the course. Moreover, the course's closed internet exam policy may further discourage students from relying on AI to complete course assignments.

4 Limitations and Future Work

While the course redesign yielded positive results across multiple instructional components, several limitations must be acknowledged when interpreting the findings. First, the use of clicker questions as a participation-based activity introduced challenges related to academic integrity. Despite policies requiring in-person attendance, some students were able to respond remotely (i.e., respond on their app without physically being in lecture attendance), thereby receiving full credit without engaging in the intended in-class environment. This behavior diminishes the accuracy of clicker data as an indicator of student engagement. A related concern is that students physically present in the classroom may still respond to questions randomly simply to obtain participation credit without being actively engaged. In such cases, the clicker responses may not reflect genuine understanding, reducing their utility as a formative diagnostic tool and limiting the instructor's ability to adapt instruction in real time.

A second limitation concerns the interpretation of student performance on low-stakes programming assignments. Although survey results suggested relatively modest reliance on AI tools, any use of AI, combined with unlimited submissions and instant feedback, complicates efforts to evaluate genuine independent mastery of programming concepts. The iterative nature of the system may incentivize trial-and-error strategies rather than reflective debugging, producing correct solutions without equivalent gains in conceptual understanding. Similarly, the effectiveness of Discussion sessions was partially dependent on the facilitation abilities of individual TAs. Variations in TA experience, communication style, and familiarity with the automated grading system may have resulted in inconsistent student experiences across quarters. Standardizing TA training for Discussion sessions may help ensure more consistent facilitation across quarters. Finally, the student survey data used for analysis were collected on a voluntary basis, creating the possibility of self-selection bias. Despite high voluntary participation in these surveys, students who were either highly satisfied or highly dissatisfied may have been more likely to respond, potentially skewing the representation of average student sentiment.

Over future quarters, several opportunities exist to further refine and extend the course redesign. One possible area of future work involves shifting clicker questions from participation-based grading to correctness-based grading. Such a change may reduce random guessing and encourage deeper engagement, though it must be implemented carefully to avoid increasing student anxiety or discouraging participation altogether (e.g., using correctness as extra credit). Another direction involves reevaluating the role of low-stakes programming assignments in an era of widespread AI availability. One pathway would be to eliminate these assignments entirely (i.e., use them as optional assignment but not factor them into the final grade) and rely solely on closed-environment assessments to ensure fair measurement of independent learning. However, given that survey data indicated limited AI dependence among students, this shift may not be necessary and could inadvertently reduce valuable opportunities for practice and feedback. A

more balanced approach may involve continued use of these assignments with enhanced guidance on reflective debugging and responsible AI use.

5 Conclusion

This paper presented a redesign of a second level programming course focused on object-oriented programming and data structures in C++ at the University of California San Diego. The redesign introduced four interventions over several quarters: redesigned student-centered lectures, redesigned programming assignments with unlimited submissions and immediate feedback, the redesign of supplemental instruction sessions and assignments, and the integration allowance of large language models (LLMs). Student feedback was collected across multiple quarters through formal course evaluations and informal surveys. Responses indicated that students valued the increased interactivity in lectures and appreciated the opportunity to resubmit assignments until mastery. Moreover, the responses further indicated that the majority of students are not relying on LLMs to complete assignments, which is consistent with the closed internet testing policy in the course. Future work will continue to refine various aspects such as consistent TA training and more emphasis on independent assessment. Moreover, future work will also investigate the long-term impact of the redesign by examining student performance in subsequent upper-division courses.

References

[1] B. M. Notaroš, R. McCullough, S. B. Manić, and A. A. Maciejewski, "Computer-assisted learning of electromagnetics through matlab programming of electromagnetic fields in the creativity thread of an integrated approach to electrical engineering education," *Computer Applications in Engineering Education*, vol. 27, no. 2, pp. 271–287, 2019.

[2] J. Kittur, "Measuring the programming self-efficacy of electrical and electronics engineering students," *IEEE Transactions on Education*, vol. 63, no. 3, pp. 216–223, 2020.

[3] P. Wong and B. Pejcinovic, "Teaching matlab and c programming in first-year electrical engineering courses using a data acquisition device," in *2015 ASEE Annual Conference & Exposition*, 2015, pp. 26–1480.

[4] P. Mayer and R. Baraniuk, "The importance of teaching logic to computer scientists and electrical engineers," *ACM Transactions on Computing Education*, vol. 25, no. 2, pp. 1–12, 2025.

[5] A. R. Bali and Y. Mughal, "The impact of different computing tools in electrical engineering learning," *Journal of Engineering and Computational Intelligence Review*, vol. 3, no. 1, pp. 52–67, 2025.

[6] J. Kittur, "Measuring the programming self-efficacy of electrical and electronics engineering students," *IEEE Transactions on Education*, vol. 63, no. 3, pp. 216–223, 2020.

[7] D.-M. Córdova-Esparza, J.-A. Romero-González, K.-E. Córdova-Esparza, J. Terven, and R.-E. López-Martínez, "Active learning strategies in computer science education: A systematic review," *Multimodal Technologies and Interaction*, vol. 8, no. 6, p. 50, 2024.

[8] J. Qadir, "Engineering education in the era of chatgpt: Promise and pitfalls of generative ai for education," in *2023 IEEE global engineering education conference (EDUCON)*. IEEE, 2023, pp. 1–9.

[9] C. Liu, G.-C. Wang, and H.-F. Wang, "The application of artificial intelligence in engineering education: A systematic review," *IEEE Access*, 2025.

[10] Y. Liu, Y. Jing, J. Li, J. Dai, Z. Hu, and C. Wang, "Application of ai in engineering education: A bibliometric study," *Review of Education*, vol. 13, no. 1, p. e70044, 2025.

[11] S. M. Vidalis and R. Subramanian, "Impact of ai tools on engineering education," in *2023 Fall Mid Atlantic Conference: Meeting our students where they are and getting them where they need to be*, 2023.

[12] T. Balart and K. J. Shryock, "A framework for integrating ai into engineering education, empowering human-centered approach for industry 5.0," in *2024 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2024, pp. 1–10.

[13] K. M. Bursic, "Does the use of clickers increase conceptual understanding in the engineering economy classroom?" in *2012 ASEE Annual Conference & Exposition*, 2012, pp. 25–479.

[14] M. V. Lubarda, A. M. Phan, A. D. Carrigg, K. Srinivasan, and J. Relaford-Doyle, "Effect of automated instantaneous feedback, unlimited submission attempts, and optional exercises on student engagement, performance, and academic integrity in an introductory computer programming course for engineers," in *2023 ASEE Annual Conference & Exposition*, 2023.

[15] B. Hanks, S. Fitzgerald, R. McCauley, L. Murphy, and C. Zander, "Pair programming in education: A literature review," *Computer Science Education*, vol. 21, no. 2, pp. 135–173, 2011.

[16] N. Raihan, M. L. Siddiq, J. C. Santos, and M. Zampieri, "Large language models in computer science education: A systematic literature review," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 2025, pp. 938–944.