The Design of an AI-Integrated, Inquiry-Based Entry-Level Computing Course for Aerospace Engineering Students

1 Introduction and Motivation

Entry-level computing courses play a critical role in shaping how engineering students understand problem solving, abstraction, and the relationship between human reasoning and computational execution. At the same time, students increasingly engage generative artificial intelligence tools outside of formal instructional guidance and are also looking to course instructors for direction on appropriate and effective usage of easily accessible generative AI tools[1]. ChatGPT and related large language models are now widely accessible and increasingly integrated into students' workflows, raising important questions about how foundational computing skills should be taught, practiced, and assessed.

The course described in this paper was not redesigned solely in response to the emergence of generative AI. Rather, it represents the first offering of a new entry-level MATLAB-based computing course that had been planned over multiple semesters as part of a broader curricular restructuring. The launch of this course provided a timely opportunity to intentionally explore how accessible generative AI tools (referred throughout the rest of the paper as the colloquial term: "AI") might be incorporated in ways that align with established learning objectives, rather than reacting through ad hoc restrictions or avoidance. The goal was not to foreground AI as the focus of the course, but to acknowledge its presence and leverage it where appropriate to support learning, reflection, assessment integrity, and a future-looking mindset.

A central challenge motivating this work is the tension between preserving conceptual rigor and accommodating the realities of AI-mediated practice. On one hand, computing courses aim to develop core competencies such as computational thinking, algorithmic reasoning, and the ability to write, test, and debug code. On the other hand, generative AI systems can now produce syntactically correct and often functional code with minimal effort from the user. This capability can obscure underlying logic and shift student attention toward output generation rather than understanding. Ignoring this tension risks leaving students unprepared to critically evaluate or meaningfully engage with AI-generated code in academic or professional contexts.

This paper adopts a systems-level perspective on AI integration within an entry-level computing course, spanning both (1) student-facing learning experiences and (2) instructor-facing assessment and feedback workflows. Rather than presenting a single narrowly scoped efficacy study, the paper documents design choices intended to preserve conceptual rigor and assessment authenticity in an AI-rich environment, and reports early descriptive signals from two course components: a structured reflection activity based on an early AI-mediated programming artifact,

and an AI-augmented feedback workflow for short-answer conceptual assessments. Each component is presented to establish feasibility and surface pedagogically relevant patterns, while deeper empirical evaluation is planned for future work.

2 Background and Related Work

Generative AI is rapidly reshaping how novices and professionals write and reason about code. Large language models trained on code can synthesize working programs from natural-language intent and are increasingly embedded into development environments, enabling interactive code generation workflows[2]. Empirical studies show that these tools shift how users allocate effort across planning, implementation, and debugging, while also introducing new failure modes such as over-trust, shallow review, and difficulty diagnosing subtle errors[3]. These dynamics are particularly consequential in entry-level computing courses, where students are still forming mental models of program execution and algorithmic structure.

Recent discourse has described these workflows as "vibe coding": AI-mediated programming characterized by high-level intent specification, rapid iteration, and reduced direct engagement with code artifacts[4]. From an educational perspective, this highlights a central tension: while AI can accelerate translation from intent to executable code, it can also bypass foundational skills unless course structures explicitly require explanation, interpretation, and verification. Learning activities that make this translation visible (e.g., converting a "recipe" or procedure into code) therefore serve as productive entry points for foregrounding ambiguity, specification quality, and the necessity of debugging and validation.

The present work draws on educational traditions that treat explanation and reflection as central evidence of understanding. Reflective prompts can reveal how learners conceptualize code, interpret program behavior, and describe changes in their thinking over time. In educational research, such reflections are commonly analyzed using qualitative methods like thematic analysis, which provide systematic approaches for identifying recurring patterns in meaning[5,6,7]. These methods are well suited to capturing student reasoning around AI, which often appears in implicit, narrative, or hedged forms that do not map cleanly onto fixed-response instruments.

In large-enrollment courses, analyzing open-ended reflections poses scalability challenges. Prior work in natural language processing has explored embedding-based similarity and, more recently, large language models to assist with qualitative coding tasks such as proposing themes, applying codebooks, and summarizing patterns[8,9,10,11,12,13]. Across these studies, a consistent recommendation is to treat LLM outputs as analytic aids rather than ground truth, preserving human oversight through transparent prompts, spot-checking, and evidence-based reporting.

Together, these strands motivate the approach used in this paper. First, student-facing AI integration is positioned as an object of inquiry: learners interact with AI to translate intent into code, then revisit and explain that code after developing foundational competence. Second, behind-the-scenes AI integration is positioned as instructional infrastructure: AI can draft feedback at scale, while humans retain responsibility for correctness, tone, and pedagogical alignment. Finally, the analysis of student reflections follows established qualitative traditions

while leveraging computational techniques (e.g., embeddings and LLM-assisted adjudication) to support scalability, with the explicit goal of maintaining interpretability and evidence-based claims.

2.1 Explanation-Centered Assessment in AI-Rich Computing Courses

The emergence of AI-mediated programming workflows such as vibe coding has important implications not only for how students generate code, but also for how instructors assess conceptual understanding. When students can easily use generative AI platforms to readily produce syntactically correct and often functional programs through high-level prompting and minimal iterative refinement, correctness of output alone becomes an increasingly weak proxy for understanding. This shift amplifies a longstanding concern in computing education: how to design assessments that surface student reasoning, interpretation, and misconceptions rather than rewarding surface-level performance.

One response to this challenge is to emphasize explanation-centered assessment formats that require students to articulate why a solution is correct or incorrect. Such approaches align naturally with instructional designs that foreground translation, interpretation, and critique of code—skills that remain essential even when AI tools are available. However, these formats also introduce practical challenges at scale, particularly in large-enrollment courses, as open-ended responses are more time-consuming to evaluate and to provide meaningful feedback on than traditional objective items. The assessment strategy used in this course builds on prior work exploring how short-answer explanations can be paired with structured prompts and AI-assisted evaluation workflows to provide timely and efficient personalized feedback to students.

Concept inventories (CIs) have been widely used in engineering education to assess students' conceptual understanding through carefully designed multiple-choice questions that target known misconceptions [14,15]. These instruments rely on expert-developed distractors to probe common patterns of incorrect reasoning and have been used extensively to characterize conceptual understanding across disciplines [16].

Rather than deploying a formal concept inventory, this course adopts the *design logic* of CIs while relaxing their measurement constraints. Weekly assessments use multiple-choice questions with CI-style distractors, but students are required to justify both a selected correct answer and an incorrect alternative using short-answer explanations. This structure preserves the diagnostic value of misconception-aware distractors while shifting the evidentiary burden from answer selection to student reasoning.

This design choice reduces reliance on psychometric standardization and instead leverages open-ended explanations to surface misconceptions directly in students' own language. Prior work has shown that explanation-centered formats, including multiple-choice or true/false questions paired with required justification, can more effectively reveal conceptual gaps than objective items alone [17,18]. In this context, short-answer explanations provide a flexible and interpretable alternative to formal CIs while remaining aligned with their underlying pedagogical intent.

In prior work presented at the redacted for review ASEE Rocky Mountain Section Conference [19],

we explored this assessment format in the context of engineering education and investigated the use of large language models to assist with generating individualized feedback on short-answer explanations. The present work extends that effort in two key ways. First, students are asked to provide short-answer responses paired with multiple-choice, concept-inventory-style questions that include distractors. Second, we characterize the AI-augmented grading workflow in terms of human grader effort and capacity to deliver personalized, student-specific feedback based on those short-answer responses.

3 Course Context and Learning Objectives

The course examined in this study was taught in Fall 2025 to approximately 100 first-year aerospace engineering students. It serves as an entry-level computing course and functions as a prerequisite for several core courses in the aerospace engineering curriculum. As such, the course occupies a foundational role, introducing students to computational thinking and programming practices that are expected to support their success in subsequent technical coursework.

The course was designed to provide students with an introduction to programming using MATLAB. MATLAB was selected at the departmental curriculum level due to its prevalence in aerospace engineering curricula, availability of discipline-aligned training resources, and integration with existing instructional infrastructure; the course design emphasizes transferable computational reasoning rather than language-specific syntax. Given the early position of the course in the curriculum, students entered with varied prior exposure to computing, ranging from no formal programming experience to limited familiarity gained through high school coursework or extracurricular activities. The instructional design therefore emphasized conceptual understanding and transferable problem-solving skills rather than proficiency in any single syntax or tool. The learning objectives for the course reflect these priorities. By the end of the semester, students were expected to explain and apply core principles of computational thinking, including decomposition of complex problems, recognition of patterns, abstraction of relevant features, and development of step-by-step algorithms. Students were also expected to implement fundamental programming constructs such as variables, control structures, arrays, and functions, and to understand how these constructs support logical program flow. In addition, the course aimed to develop students' ability to create, test, and debug MATLAB programs used to solve basic aerospace engineering application problems, reinforcing the connection between computation and disciplinary context.

These objectives were intentionally framed to emphasize reasoning, interpretation, and debugging rather than code production alone. The instructional and assessment strategies described in subsequent sections were designed to support these objectives while also accounting for the increasing accessibility of generative AI tools. AI integration was treated as a contextual consideration layered onto these established goals, rather than as a redefinition of what students were expected to learn.

4 Design Principles for AI-Integrated Computing Instruction

Instructional and assessment choices were guided by a small set of design principles intended to preserve the core learning objectives of an entry-level computing course while acknowledging the

growing presence of generative AI tools in students' academic and professional lives. AI tools were incorporated selectively where they could support, reveal, or stress-test student understanding, but not as a substitute for it.

First, *Reasoning over Syntax* frames programming as explicit logical reasoning rather than memorization of language-specific constructs. Activities were designed to make the mapping between natural language intent and executable code visible, emphasizing decomposition, abstraction, and algorithmic flow. In this framing, AI can act as a translation mechanism that exposes both the power and limitations of moving from informal intent to formal code.

Second, *Explanation as Evidence* treats the ability to read, explain, and critique code as a primary indicator of computational understanding. Assessments were therefore designed to prioritize articulation of understanding (e.g., reasoning about program behavior, identifying errors, and justifying design choices) rather than output generation alone.

Third, *Authentic Assessment* emphasizes assessment formats that remain meaningful in an environment where AI tools are readily available. Rather than relying primarily on take-home code submissions, the course included in-person explanation-based assessments and short-answer reasoning prompts that require student understanding.

Finally, *Scalable Feedback with Oversight* acknowledges the feasibility constraints of large-enrollment instruction and positions AI as a evaluation aid to improve feedback timeliness and specificity while preserving human oversight. In this paper, this principle is operationalized through an AI-augmented workflow for short-answer feedback, described descriptively to establish feasibility and characterize grader interactions.

These design principles provided a common framework for structuring instruction and assessment across the semester. The following sections describe the primary instructional and assessment interventions, organized to make explicit how each component enacts these principles in practice and how AI was integrated both student-facing and behind the scenes.

5 Instructional and Assessment Interventions

This section describes the major instructional and assessment components of the course and how each component enacts the design principles introduced in the previous section. The intent was not to maximize AI usage, but to integrate it selectively in ways that reinforced core computing learning objectives.

For clarity, the design principles are summarized here using concise descriptors referenced throughout the remainder of the paper: *Reasoning over Syntax*, *Explanation as Evidence*, *Authentic Assessment*, and *Scalable Feedback with Oversight*. Table 1 provides a high-level mapping between the design principles and major course components.

5.1 Early-Semester AI "Recipe" Laboratory

The first laboratory activity was designed as an early-semester, low-stakes introduction to algorithms, programming, and generative AI. Rather than beginning with syntax, the lab asked students to reason through a familiar process: calculating their final course grade. Because the

Table 1: Mapping of design principles to instructional and assessment interventions

Course Component	Reasoning over Syntax	Explanation as Evidence	Authentic Assessment	Scalable Feedback
AI "Recipe" Laboratory	X	X		
Code Interviews (Provided Code)		X	X	
Code Interviews (Student Code)		X	X	
Short-Answer Concept Questions		X	X	X
AI-Augmented Grading Pipeline				X
MATLAB Academy Integration	X			
Final Project	X	X	X	

course used a tiered, mastery-based grading structure unfamiliar to many students, this activity also served as an applied way to engage with the grading policy.

Students first wrote a detailed, step-by-step description in plain language of how a program could determine a final letter grade based on syllabus criteria. They were prompted to identify inputs, operations, and outputs, but were told not to write code. After completing their natural language "recipe," students used a generative AI tool to translate the description into a MATLAB script using introductory constructs. Students were not required but many decided to run the code with sample inputs and began annotating it, noting confusion and asking the AI or members of the teaching to explain specific segments.

Later in the semester, after students had gained MATLAB fluency, they revisited the AI-generated code, annotated it fully in their own words, and validated correctness using provided test cases. Students then submitted a short reflection describing how their understanding evolved from initial co-generation to later interpretation. These reflections constitute the primary data source for the reflection analysis reported in this paper.

Additional details related to the grading structure, validation cases, and AI resources provided to students are included in Appendix A.

5.2 Code Interviews on Provided and Student-Generated Code

In-person code interviews were used as a primary assessment mechanism to evaluate conceptual understanding of programming fundamentals in ways that remain meaningful in an AI-rich environment. Interviews required students to articulate their understanding of code behavior and underlying logic in real time, emphasizing explanation and reasoning over polished code artifacts.

The first code interview focused on provided code as a means of assessing students' conceptual understanding while emphasizing a critical skill: the ability to read and reason about code written by others. Reading and reasoning about code is foundational for debugging, and the ability to do so with code authored by others is increasingly important in an era where code may originate from generative AI systems. Students were given a printed MATLAB program and approximately seven minutes to review and make notes, followed by an interview of similar length with a

member of the teaching team. Interview questions probed core concepts such as control flow, variable usage, and conditional logic and increased in complexity to distinguish surface familiarity from deeper understanding. The second code interview occurred later in the semester and focused on student-generated code from a course assignment. Because students were explaining their own programs, the interview emphasized high-level structure, design decisions, and how program components worked together.

While code interviews were resource intensive, informal observations suggested they provided immediate and actionable feedback, particularly for students who were not yet conceptually on track. Pass rates improved substantially from the first to the second interview, consistent with increased familiarity with expectations and greater conceptual grounding. An optional AI-based practice chatbot hosted on external infrastructure was made available to support interview preparation; its use was not required and is not a central focus of the present study.

5.3 Short-Answer Concept Questions with AI-Augmented Grading

Weekly concept questions were used throughout the semester to assess students' understanding of foundational computing ideas. These questions used a multiple-choice format with five options paired with a required short-answer explanation. Students completed: "The correct answer is ___ because . . . and one incorrect answer is ___ because . . . ". This structure combines diagnostic distractors with written reasoning, emphasizing conceptual discrimination rather than answer selection alone.

The administration and high-level architecture builds on prior work presented at a regional ASEE conference in Spring 2025 redacted citation for review[19]. In the present course, the workflow was revised to incorporate structured human oversight prior to returning feedback to students. After students completed quizzes in the learning management system, responses were exported and de-identified for compliance. An AI-based evaluation pipeline using OpenAI GPT-4.1-mini[20] drafted feedback relative to an instructor-provided expert solution, and instructional staff reviewed and edited this feedback before release.

Students were explicitly informed that an AI-augmented grading pipeline was used and were encouraged to read their feedback critically and to bring any potential errors to the teaching team. No students contacted the instructional team to report suspected errors in released feedback.

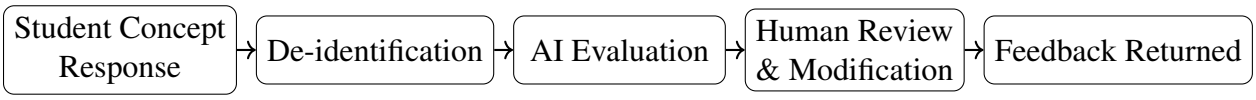

Figure 1: Simplified overview of the AI-augmented grading pipeline for short-answer concept questions. Student responses are de-identified, evaluated by an AI system, reviewed by a human grader, and then returned to students with feedback.

5.4 MATLAB Academy Integration and Self-Guided Learning

MATLAB Academy modules were integrated as asynchronous homework aligned with lecture topics and in-class activities. These modules provided structured, self-paced instruction on core

programming concepts and MATLAB functionality, freeing in-person time for discussion, debugging, and conceptual clarification. An additional motivation was to encourage students to pursue professional-grade credentials and certificates that extend beyond a traditional four-year undergraduate education. This credentialing structure was framed as a professional development opportunity and as a mechanism for encouraging self-guided learning.

Integration was facilitated through synchronization with Canvas, allowing instructional staff to monitor student progress without manual collection or grading. Course completion was aligned with externally recognized MATLAB Academy training and digital credentials[21,22], with additional optional modules available for students pursuing higher levels of proficiency[23,24,25,26,27].

5.5 Final Project and AI-Supported Scaffolding

The final project served as a culminating experience in which students designed and implemented a MATLAB program of their choosing over approximately four calendar weeks. Projects were required to include core programming elements (e.g., user-defined functions, control structures, arrays or matrices, input/output operations, and at least one visualization) to encourage thoughtful program structure.

The project permitted the use of generative AI tools during development with the expectation that students could explain and justify all aspects of their code. Students submitted completed code, a short presentation, and a portfolio-style summary page. During a project symposium, students presented their project and participated in a brief code-interview describing their design and implementation.

6 Methods: Analysis of Student Reflection Responses

This study examines student reflections associated with the early-semester AI "recipe" laboratory described in Section 5.1. The purpose of the analysis was to understand how students conceptualized programming and generative AI after co-generating code early in the semester and revisiting that artifact later with additional technical knowledge. The methods described here focus exclusively on these reflections and do not attempt to evaluate learning outcomes across the entire course.

6.1 Context and Data Source

The data consist of written reflections submitted by students enrolled in an introductory computing course for aerospace engineering majors during the Fall 2025 semester ($\sim$100 students). At the course conclusion, students submitted a one-page free form reflection (approximately 250 words) after revisiting AI-generated MATLAB code later in the semester. The reflection prompt asked students to address four guiding questions related to (1) writing a natural-language algorithm, (2) observing AI translation into MATLAB logic, (3) reassessing the code after learning MATLAB, and (4) reflecting on how their perspective on generative AI changed over time.

Reflections were submitted through the learning management system as part of a graded assignment and were graded for completion and thoughtfulness rather than correctness or position. Prior to analysis, reflections were exported and de-identified to remove student names, identification numbers, and other potentially identifying information. The analysis was conducted on anonymized text only. An IRB exempt letter was obtained prior to engaging with the analysis of the student reflections and the study was conducted in alignment with the IRB exempt letter.

6.2 Analytical Goals

The analysis was designed to address: What themes emerge in students' reflections about the relationship between natural language reasoning and code structure? How do students describe their evolving understanding of AI-generated code over time? How do students characterize the role of generative AI in their learning by the end of the semester?

The study adopted an exploratory, qualitative approach aimed at surfacing recurring patterns in student thinking. Because reflections were collected from a large-enrollment course and written in free-form style, the analysis emphasized interpretable thematic patterns and scalable processes rather than hypothesis testing.

6.3 AI-Assisted Thematic Analysis with Human Oversight

A human-in-the-loop, LLM-assisted thematic analysis approach was used to balance interpretability with scalability. For each reflection prompt, five student submissions were randomly sampled and provided to a language model to propose an initial thematic codebook of 3-5 themes intended to capture dominant patterns in student reasoning. This initial codebook was treated as provisional.

The codebook was then iteratively refined using additional small subsets of student responses. In each refinement round, the language model was tasked with (1) judging whether each response aligned with one or more existing themes and (2) flagging cases where the response did not fit the current codebook. When such mismatches occurred, the model proposed either a new theme or a targeted modification (e.g., rewording, splitting, or merging themes) intended to improve coverage and reduce overlap. Importantly, proposed refinements were not automatically adopted. The research team spot-checked flagged responses and the model's suggested changes, accepting only those refinements judged to improve coherence and represent distinct ideas present in the data. This iterative process continued until additional rounds produced minimal changes and the codebook stabilized for each prompt, resulting in a short definition and representative examples for each theme.

To estimate theme prevalence across the full cohort, a language model was then used to evaluate each student response against the finalized, prompt-specific codebook. The model was instructed to assign a similarity score between zero and one for each theme and to provide a short excerpt from the student response as evidence when a theme was judged to be present. If no themes were supported by the text, the model returned a standardized "missing" label. Outputs were validated for consistent structure and completeness, and periodic spot checks of randomly selected

responses were used to verify that extracted evidence and theme assignments were reasonable.

As a secondary validation step, the full set of de-identified reflections was analyzed using NotebookLM[28] as a document-level synthesis tool. Without applying a predefined codebook, NotebookLM independently identified common themes across the corpus, which showed strong qualitative alignment with those produced through the iterative, LLM-assisted codebook development process.

6.4 Limitations of the Methodological Approach

This analysis relies on student self-reported reflections, which capture perceptions rather than direct measures of learning. Human judgment remains embedded in key stages (e.g., codebook development and prompt design), and the use of large language models introduces additional sources of uncertainty related to model behavior and prompt sensitivity. Random samples of reflections and results were observed by a human evaluator for correctness throughout the evaluation process, but systematic interrater validation is beyond the scope of this conference paper.

7 Results - Condensed Thematic Prevalence Across Reflection Prompts

This section presents descriptive results from thematic analysis of student reflections associated with the introductory "recipe" laboratory. The reflection prompts asked students to articulate how their understanding of code, AI translation, and their own learning evolved over the course of the semester.

Table 2 summarizes dominant themes for each reflection prompt using the evidence-constrained LLM adjudication. Percentages indicate the proportion of students exhibiting strong alignment with each theme. Only the top themes per question are shown to highlight dominant patterns.

Across prompts, student reflections consistently emphasized programming as a process of structured reasoning and interpretation rather than syntax alone. Writing a natural-language recipe and later revisiting AI-generated code helped many students articulate logical flow, recognize sources of earlier confusion, and develop greater appreciation for program structure and clarity. The strongest alignment emerged in reflections on generative AI use, where most students described a shift toward viewing AI as a useful but imperfect learning aid that requires active understanding and critical oversight.

Taken together, reflection results suggest that early exposure to AI-mediated code generation, coupled with later revisitation and explanation-based expectations, supported a more nuanced student view of both programming and AI. These reflections provide descriptive evidence that students can develop metacognitive awareness about AI use and recognize the centrality of reasoning and interpretation in programming, even in an AI-rich environment.

Table 2: Condensed summary of dominant themes identified in student reflections. Percentages indicate the proportion of students exhibiting strong alignment with each theme.

Question	Dominant Theme	% Alignment
How did writing the grade calculation recipe in plain language shape your understanding of how code works?	Understanding code logic and flow	67%
	Step-by-step logical thinking	61%
What surprised you most about how the AI translated your words into MATLAB logic?	Surprise at AI accuracy and efficiency	64%
	Appreciation of code simplicity and clarity	41%
Looking back now, what parts of your early code make more sense or less sense, now that you've learned how to code?	Recognition of code simplicity and clarity	73%
	Identification of confusion or complexity	57%
How has your perspective on using GenAI for learning changed over the course of the semester?	Importance of user understanding and engagement	92%
	GenAI as a useful but imperfect tool	83%
	Shift from viewing GenAI as a shortcut to a learning aid	64%

8 Results: AI-Augmented Grading Practices

This section presents grader behavior in the AI-augmented feedback workflow, focusing on review efficiency, the nature of human intervention relative to AI-generated feedback, and overall feedback throughput. Results are presented descriptively to document how graders interacted with AI-generated feedback in an authentic course setting, rather than to compare against a non-AI baseline.

Table 3 summarizes inferred grading efficiency by grader. Across all graders, median grading times per submission were well under one minute, ranging from 0.44 to 0.91 minutes. Mean grading times were consistently higher than medians, reflecting long-tailed distributions with occasional higher-effort cases. Variability across graders was observed, indicating that AI augmentation did not homogenize grading behavior.

Table 4 characterizes the overall nature of feedback released to students relative to the AI-generated feedback, including both unchanged and modified cases. For four of five graders, approximately 75-80% of submissions were released unchanged, suggesting that AI drafts frequently met an acceptable standard with rapid human validation. When modifications occurred, they ranged from light editorial changes to less frequent substantive rewrites. One grader exhibited a higher propensity to modify feedback, illustrating grader-specific differences in review style and quality thresholds. Edit size (small vs. large) was determined using absolute word-count differences between AI-generated and released feedback. Semantic change was

Table 3: Inferred grading effort per submission by grader. Times are reported in minutes.

Grader	N	Median	IQR	Mean	Std. Dev.
Grader 1	155	0.64	0.93	0.96	0.91
Grader 2	188	0.44	0.83	0.85	1.01
Grader 3	156	0.49	0.89	0.97	1.15
Grader 4	374	0.57	0.71	1.13	1.80
Grader 5	96	0.91	1.62	1.68	2.19

quantified using embedding-based cosine similarity between the two versions, with semantic change defined as one minus cosine similarity. For each grader, thresholds separating small versus large changes in both size and meaning were defined using the median values computed over that grader's modified submissions. These categories are used solely for descriptive characterization of grader behavior rather than formal statistical inference.

Table 4: Overall nature of feedback released to students relative to AI-generated feedback. Percentages are computed over all submissions for each grader.

Grader	Unchanged	Small / Small	Large / Small	Small / Large	Large / Large
Grader 1	74.9	7.1	5.5	5.5	7.1
Grader 2	81.1	6.0	3.5	3.5	6.0
Grader 3	78.0	8.3	3.0	4.8	6.0
Grader 4	77.1	8.0	3.5	3.8	7.5
Grader 5	51.9	13.5	10.6	11.5	12.5

"Small/Large" refers to edit size (absolute word-count change) and semantic change, respectively.

Table 5 reports feedback throughput, defined as the number of words of released feedback per minute of inferred grading time. Across all graders and 969 submissions, median throughput was approximately 84 words per grader-minute, with individual graders ranging from about 52 to 102 words per minute. These values characterize the capacity of the AI-augmented workflow to support substantial amounts of personalized feedback within limited review time, with human effort concentrated on quality control and targeted refinement rather than composing feedback from scratch.

In summary, the AI-augmented grading workflow points to three primary observations. First, approximately three-quarters of AI-generated evaluations were released without modification, indicating that the system frequently produced satisfactory first-pass feedback. Second, effective feedback throughput, measured as words of released feedback per minute of grader review time, was comparable to the upper range of typical human typing speeds, suggesting that grader efficiency was largely tied to review rather than text composition. Third, the substantial variability in grading times reflects a workflow in which roughly one-quarter of submissions, where AI-generated feedback was unsatisfactory, accounted for a disproportionate share of grader time on task.

Table 5: Feedback throughput by grader, measured as words of released feedback per minute of inferred grading time.

Grader	N	Median (wpm)	Mean (wpm)	Std. Dev.
Grader 1	155	68.3	100.2	89.9
Grader 2	188	102.1	146.2	174.1
Grader 3	156	92.2	109.9	82.3
Grader 4	374	86.7	99.2	78.9
Grader 5	96	52.3	77.7	69.7
Overall	969	84.3	108.1	107.3

9 Discussion

This paper presents a single-semester case study of AI integration in a large, entry-level computing course that spans both student-facing learning experiences and instructor-facing assessment workflows. The course illustrates how these roles of AI can be designed coherently within a single instructional ecosystem while preserving conceptual rigor and assessment authenticity.

On the student-facing side, reflection findings suggest that early AI-mediated code generation can be pedagogically productive when paired with later revisitation and explanation-based expectations. Students frequently described programming as structured reasoning and communication, and many reported increased ability to interpret and critique code over time. Importantly, reflections indicate that students developed more nuanced views of AI use, emphasizing that AI is helpful when paired with active understanding and critical oversight rather than treated as an answer generator.

On the instructor-facing side, descriptive results from the AI-augmented grading workflow suggest a feasible model of "scalable personalized feedback with oversight." AI-generated feedback often functioned as an acceptable first-pass draft, with graders primarily acting as reviewers who validate, adjust, and occasionally rewrite feedback. Patterns of unchanged release alongside targeted edits indicate that AI can shift human effort from composing feedback to quality control and pedagogical tuning. The observed variability across graders also suggests that AI augmentation can accommodate individual grading styles rather than forcing uniform behavior.

Taken together, these findings support the broader claim that AI integration in computing education is quite productive when it is paired with instructional structures that foreground reasoning and explanation, and human oversight remains critical in assessment and feedback. As generative AI becomes increasingly common in engineering contexts, design-principled integration across both learning and instructional infrastructure may be a practical path to preparing students for AI-mediated practice without compromising foundational educational goals.

10 Limitations and Future Work

This study has several limitations that constrain the scope of its conclusions. First, findings are drawn from a single offering of an entry-level computing course at one institution with a specific student population and instructional context. Second, the primary student-facing evidence is based on self-reported reflections, which capture perceptions and metacognition rather than direct measures of learning or performance. Third, AI-assisted thematic analysis introduces uncertainty related to model behavior and prompt sensitivity; while periodic human evaluation spot checks were used along with an additional independent evaluation to validate reasonableness, more systematic validation and a larger sample size are needed before substantial conclusions can be made.

The grading workflow results are also descriptive and do not include a non-AI comparison condition. While the results characterize how graders interacted with AI-generated drafts, future work is needed to evaluate how prompt design, grader training, and workflow variations influence both feedback quality and student uptake. More broadly, future studies will examine longitudinal impacts, including whether early emphasis on explanation, interpretation, and critical AI use influences subsequent performance in upper-division aerospace engineering coursework. Additional work will also explore tighter triangulation between reflections and other assessment artifacts (e.g., code interview outcomes and project evaluations) and incorporate more systematic validation of AI-assisted personalized feedback workflows.

Finally, this conference paper intentionally sets the stage for deeper, more distinct future studies: one focused on student-facing AI-integrated learning design and outcomes, and one focused on instructor-facing AI-augmented assessment and personalized feedback workflows. Both threads warrant expanded empirical evaluation beyond the descriptive scope presented here.

11 Conclusion

This paper presents a principled, systems-level approach to integrating generative AI into an entry-level computing course that was already undergoing curricular renewal, rather than being redesigned in response to AI alone. By grounding instructional and assessment choices in explicit design principles, the course aimed to preserve conceptual rigor, emphasize explanation and interpretation, and maintain assessment authenticity in an AI-rich environment. The interventions described illustrate how AI can be incorporated both as a student-facing learning catalyst and as an instructor-facing feedback amplifier, while keeping human reasoning and oversight central.

Early descriptive evidence from student reflections suggests that structured AI exposure paired with later revisitation can support students in viewing programming as reasoning and communication and in developing more critical, nuanced perspectives on AI use. Evidence from the grading workflow suggests that AI-generated drafts can support scalable, personalized feedback with targeted human review rather than full automation. Together, these results provide a coherent example of how AI can be integrated across the course lifecycle in ways that align with enduring computing learning objectives, and motivate deeper future work focused separately on student learning outcomes and assessment infrastructure.

References

[1] Center for Teaching & Learning, University of Colorado Boulder. Results from the undergraduate perspectives on ai at cu boulder survey. https://www.colorado.edu/center/teaching-learning/programs-services/undergraduate-student-programs/educational-technology-research-assistants-etra-0, 2025. Accessed 2026-01-19.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[3] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI EA '22)*. ACM, 2022. doi: 10.1145/3491101.3519665.

[4] Advait Sarkar and Ian Drosos. Vibe coding: programming through conversation with artificial intelligence. Preprint, 2025. URL https://advait.org/files/sarkar_2025_vibe_coding.pdf. Accessed 2026-01-17.

[5] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006. doi: 10.1191/1478088706qp063oa.

[6] Johnny Saldaña. *The Coding Manual for Qualitative Researchers*. SAGE Publications, 4 edition, 2021.

[7] Matthew B. Miles, A. Michael Huberman, and Johnny Saldaña. *Qualitative Data Analysis: A Methods Sourcebook*. SAGE Publications, 3 edition, 2014.

[8] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, 2019. URL https://arxiv.org/abs/1908.10084.

[9] Austin C. Kozlowski, Matt Taddy, and James A. Evans. The geometry of culture: Analyzing the meanings of class through word embeddings. *Proceedings of the National Academy of Sciences*, 116(37):18322–18328, 2019. doi: 10.1073/pnas.1903858116.

[10] David L. Morgan. Exploring the use of artificial intelligence for qualitative data analysis: The case of chatgpt. *International Journal of Qualitative Methods*, 22:1–10, 2023. doi: 10.1177/16094069231211248.

[11] Xiner Liu, Adrian Zambrano, Ryan S. Baker, Alex Barany, Jaclyn Ocumpaugh, Junyi Zhang, David Pankiewicz, Rami Nasiar, and Jasper Wei. Qualitative coding with gpt-4: Where it works better. *Journal of Learning Analytics*, 12(1), 2025. doi: 10.18608/jla.2025.9112.

[12] Xiner Liu, Junyi Zhang, Alex Barany, Ryan S. Baker, et al. Assessing the potential and limits of large language models in qualitative coding. In *Proceedings of the International Conference on Quantitative Ethnography*. Springer, 2024. doi: 10.1007/978-3-031-76335-9_7.

[13] He Zhang, Chuhao Wu, Jingyi Xie, ChanMin Kim, and John M. Carroll. Qualigpt: Gpt as an easy-to-use tool for qualitative coding. *arXiv preprint arXiv:2310.07061*, 2023. URL https://arxiv.org/abs/2310.07061.

[14] David Hestenes, Malcolm Wells, Gregg Swackhamer, et al. Force concept inventory. *The physics teacher*, 30 (3):141–158, 1992.

[15] Julie Libarkin. Concept inventories in higher education science, 2008. URL https://sites.nationalacademies.org/cs/groups/dbassesite/documents/webpage/dbasse_072624.pdf.

[16] David Hestenes and Ibrahim Halloun. Interpreting the force concept inventory. *The Physics Teacher*, 33(8): 504–506, 1995.

[17] Amy G Briggs, Lee E Hughes, Robert E Brennan, John Buchner, Rachel EA Horak, D Sue Katz Amburn, Ann H McDonald, Todd P Primm, Ann C Smith, Ann M Stevens, et al. Concept inventory development reveals common student misconceptions about microbiology. *Journal of Microbiology & Biology Education*, 18(3): 10–1128, 2017.

[18] Tony Wright and Susan Hamilton. Assessing student understanding in the molecular life sciences using a concept inventory. *ATN Assessment*, 2008.

[19] Chloe J. Long and Bobby Hodgkinson. Ai-driven concept assessment framework and application in aerospace engineering sophomore lab. In *Proceedings of the 2025 ASEE Rocky Mountain Section Conference*, Boulder, Colorado, May 2025. American Society for Engineering Education. ASEE Rocky Mountain Section Conference, University of Colorado Boulder.

[20] OpenAI. Gpt-4.1-mini. https://platform.openai.com/docs/models, 2024. Accessed 2026-01-19.

[21] MathWorks. Core matlab skills. https://matlabacademy.mathworks.com/details/core-matlab-skills/lpmlcms, 2024. MATLAB Academy digital credential.

[22] MathWorks. Programming in matlab. https://matlabacademy.mathworks.com/details/programming-in-matlab/lpmlprm, 2024. MATLAB Academy digital credential.

[23] MathWorks. Build matlab proficiency. https://matlabacademy.mathworks.com/details/build-matlab-proficiency/lpmlbmp, 2024. MATLAB Academy digital credential.

[24] MathWorks. Matlab skills for simulink modeling. https://matlabacademy.mathworks.com/details/matlab-skills-for-simulink-modeling/lpmlssm, 2024. MATLAB Academy digital credential.

[25] MathWorks. Data analysis in matlab. https://matlabacademy.mathworks.com/details/data-analysis-in-matlab/lpmldam, 2024. MATLAB Academy digital credential.

[26] MathWorks. Visualization in matlab. https://matlabacademy.mathworks.com/details/visualization-in-matlab/lpmlvzm, 2024. MATLAB Academy digital credential.

[27] MathWorks. Matlab for the mathworks certified matlab associate exam. https://matlabacademy.mathworks.com/details/matlab-for-the-mathworks-certified-matlab-associate-exam/lpmlmace, 2024. MATLAB Academy certification preparation.

[28] Google. Notebooklm, 2024. URL https://research.google/notebooklm/.

A Supplementary Course Context and Assignment Details

A.1 Course Grading Structure

The course employed a tiered, mastery-based grading structure designed to emphasize growth, conceptual understanding, and sustained engagement. Rather than relying on a traditional weighted-average grading scheme, final letter grades were determined by meeting a set of clearly defined requirements across multiple dimensions of the course.

All students were required to pass core laboratory assignments, complete weekly participation activities, and achieve a minimum level of conceptual understanding as measured by short-answer assessments. Higher letter grades required completion of additional technical work, including advanced MATLAB training modules and, for the highest tier, a more complex personal project. This structure was intended to encourage autonomy and self-directed learning while making expectations for each grade explicit.

Because this grading approach differed from what many first-year students had previously encountered, the structure itself became a focal point for early course activities, including the first laboratory assignment described in the main text.

A.2 MATLAB Training Requirements by Grade Tier

As part of the mastery-based structure, students completed a sequence of MATLAB Academy training modules aligned with course outcomes. At a minimum, all students who passed the course earned digital credentials for Core MATLAB Skills and Programming in MATLAB. These credentials verified foundational proficiency in MATLAB programming concepts.

Students pursuing higher letter grades completed additional training modules covering topics such as MATLAB proficiency, Simulink modeling, data analysis, and visualization. Students earning the highest grade tier also completed an advanced training module or certification preparation. Completion of these modules resulted in multiple digital credentials that students could add to professional portfolios and resumes.

A.3 Validation Cases for the Grade Calculation Laboratory

To support testing and validation of the grade calculation scripts developed in the early-semester laboratory, students were provided with a set of representative test cases. Each case specified a hypothetical student profile, including laboratory completion status, assessment averages, participation levels, training completion, and project status. Students were expected to verify that their program correctly classified each case into the appropriate final grade tier.

These validation cases served two purposes. First, they reinforced the importance of testing and debugging as part of program development. Second, they provided a concrete reference for students to reason about conditional logic and algorithmic flow in the context of a real grading policy.

A.4 AI Resources Provided to Students

Students were provided with access to several AI-based tools for MATLAB assistance, including AI systems developed or supported by MathWorks. These tools were presented as optional learning supports intended to help with code explanation, debugging, and translation of natural language descriptions into MATLAB syntax.

Students were reminded that while these tools generally follow MATLAB best practices, they are not authoritative sources and may produce incorrect or incomplete responses. Use of AI tools was permitted throughout the course, provided students could demonstrate understanding of any AI-assisted work during assessments.

A.5 Reflection Prompt for Laboratory 1

Following completion of the early-semester laboratory and a later revisiting of the AI-generated code, students submitted a written reflection of approximately 250 words. The reflection asked students to address the following prompts:

How did writing the grade calculation recipe in plain language shape your understanding of how code works?

What surprised you most about how the AI translated your words into MATLAB logic?

Looking back later in the semester, what parts of your early code made more sense or less sense after learning MATLAB?

How did your perspective on using generative AI for learning change over the course of the semester?

These reflections form the primary data source for the analysis presented in this paper.