2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #52698

Fostering Comprehension in Introductory Programming through Code Reviews and Verbal Explanations in the Age of Generative AI: An MSI Case Study

Edward Dillon, University of Maryland Baltimore County

Edward Dillon is an Associate Professor in the Department of Information System at the University of Maryland, Baltimore County (UMBC). Edward received his B.A. in Computer and Informational Science from the University of Mississippi in 2007. He would go on to obtain his Masters and Ph.D. in Computer Science from the University of Alabama in 2009 and 2012, respectively. Prior to his arrival to UMBC in August 2024, Dr. Dillon served as a Computer Science Instructor at Jackson State University (2012-2013), and a Postdoctoral Researcher at Clemson University (2013-2014) and the University of Florida (2014-2016), and Assistant/Associate Professor in the Department of Computer Science at Morgan State University (2017-2024). His research focuses on human-centered computing with emphasis on computing education, broadening participation in computing, and tech workforce prep.

Uzma Hasan, University of Maryland Baltimore County
Omobolanle Niyi Owoeye, University of Maryland Baltimore County
Kevin Lemus, University of Maryland Baltimore County

Kevin Lemus is a graduate assistant researcher and Master's student at the University of Maryland, Baltimore County (UMBC). Kevin earned his Bachelor of Science in Information Systems along with a Certificate in User Experience, Web, and Mobile Development from UMBC in 2024. He is currently advancing his expertise as a candidate for a Master of Science in Human-Centered Computing, with an expected completion date of December 2026.

SRUSHTI RAJESH DHARMALE, University of Maryland Baltimore County
William Parham, University of Maryland Baltimore County

Fostering Comprehension in Introductory Programming through Code Reviews and Verbal Explanations in the Age of Generative AI: An MSI Case Study

Abstract

Developing effective strategies to support student learning and computational proficiency in introductory programming courses has long posed a challenge. Prior research underscores the importance of program comprehension and the ability to clearly articulate one's understanding as foundational to developing programming skills. These competencies are crucial not only for success as students progress through a computing curriculum but also for their long-term professional growth.

The rapid emergence and increasing accessibility of generative artificial intelligence (GenAI) tools introduce new complexities for educators and learners alike. While these tools allow students to use large language model (LLM)-based queries to generate functional programming solutions with minimal effort, they also raise concerns about overreliance, which may hinder students' ability to independently construct and comprehend computational solutions. As GenAI tools become more pervasive in educational settings, it is essential to ensure that students continue to develop and retain proficient computational understanding rather than circumventing it through automated assistance.

This case study examines an instructional intervention implemented in an introductory programming course at a minority-serving institution (MSI) in the Mid-Atlantic United States during the Fall 2024 semester. The intervention integrated the use of the GenAI tool, ChatGPT, within selected take-home assignments and required students to participate in structured code reviews and verbal think-aloud activities. The objective was to assess students' ability to apply their programming skills and concepts while having direct access to a GenAI tool. This approach also encouraged students to articulate their understanding/comprehension of programming concepts and explain the reasoning behind their code. Additionally, the study aimed to provide initial insight on whether students could continue to develop and showcase their programming skills without becoming overreliant on GenAI tools.

A mixed methods approach was used to evaluate student performance on these selected assignments, while observing their experiences with ChatGPT. Findings indicated that most students demonstrated proficiency in interpreting code and articulating their solutions, both in writing and verbally, even while using ChatGPT. Students also reported that ChatGPT aided their understanding of programming concepts and supported their development of essential skills in code review and evaluation. This study examines how such interventions can support students'

development of program comprehension skills in a GenAI-enabled learning environment, providing insights for educators aiming to adapt instruction while maintaining rigor and depth in foundational computing education.

1. Introduction

Generative artificial intelligence (GenAI) has become increasingly prevalent in education, with its influence, use, and integration expanding at a rapid pace. However, in the context of computing education, tools like ChatGPT also raise significant pedagogical concerns. Educators face mounting challenges in ensuring that in-class assessments accurately measure students' genuine understanding and knowledge acquisition, especially given the ease with which students can use GenAI to simulate solutions. As a result, it can be difficult to determine whether students are truly mastering key concepts or simply replicating AI-generated responses.

While GenAI tools like ChatGPT can support students in early programming courses by helping them generate syntactically correct code and overcome technical barriers, their use also poses potential risks to students' cognitive development and understanding of core computational practices. Relying heavily on such tools may lead students to bypass foundational skills such as algorithmic thinking, abstraction, and debugging, as they increasingly accept ready-made AI-generated solutions, often without fully comprehending how or why the code works.

Program comprehension is widely recognized as a critical component of the computational learning process for novice programmers [1]. Beyond academic settings, this skill is also highly valued by hiring managers, as students are expected to demonstrate their understanding of code when seeking internships and job opportunities in the tech industry [2, 3]. As such, early programming courses have long emphasized the importance of helping students develop the cognitive models needed to understand structured logic and code behavior [4, 5]. These models enable students not only to construct coding solutions but also to articulate their grasp of fundamental programming concepts, an ability that remains essential even in the current era of accessible GenAI tools.

This article presents an instructional approach designed to address the challenges of fostering computational learning amid the growing accessibility of generative AI. The model was implemented in a 17-week introductory Java course at a minority-serving institution in the Mid-Atlantic United States during the Fall 2024 semester. Rather than prohibiting the use of ChatGPT, the featured GenAI tool for this initiative, the course purposely integrated it for direct use into three assignment artifacts completed by 32 students. The aim was to preserve and promote program comprehension and computational thinking while maintaining academic accountability. To assess students' understanding, the assignments incorporated structured interventions such as code reviews and verbal explanations that required students to articulate their reasoning and demonstrate a deeper grasp of the course material beyond simply submitting a written solution for a grade.

The core motivation is to prompt students to actively demonstrate and articulate their understanding of programming concepts and their written code, while alleviating the potential for superficial engagement and over-reliance with AI-generated output. This approach seeks to empower students to become active agents in their learning and not just reliant consumers of

automated solutions. From an educator's perspective, the outcomes of this work can help instructors assess students' authentic understanding by moving beyond written code submissions and incorporating verbal walk-throughs that demonstrate comprehension, clarity, logic, and reasoning. When students can effectively articulate how their code works, they demonstrate learning proficiencies that cannot be manufactured.

The goal of this work is to contribute evidence-based strategies that support and safeguard authentic computational learning in the presence of GenAI tools. By shifting the focus from correctness to explanation, this approach safeguards against superficial engagement and reinforces the students' ability to take ownership of their solutions and acquired knowledge. In the age of GenAI, where code can be copied without direct evidence of comprehension and understanding, these verbal walkthroughs serve as robust indicators of genuine learning. On a broader scale, this work also seeks to offer an evidence-backed response to the growing accessibility of GenAI tools in computing education. It promotes student agency, strengthens conceptual understanding, and redefines assessment practices in a way that is ethically responsible while maintaining pedagogical soundness.

2. Literature Review

2.1 Program Comprehension

For some time, literature has emphasized the importance of comprehension as a foundational component of novice programmers' computational skill development [6, 7, 4]. Researchers have proposed various approaches to assess and examine programming comprehension, including pedagogical code reviews [8, 9], mental model evaluations [4, 5], eye-tracking studies [10, 11, 12], and cognitive learning assessments [13, 14], among others. The literature also suggests that novice programmers possess a cognitive framework for understanding code that differs significantly from that of expert programmers [1]. As such, it has been argued that novices need to develop a more detailed and structured cognitive framework to effectively comprehend and reason through written programs [15].

Prior to the emergence of GenAI tools such as ChatGPT, comprehension had been firmly established in the literature as a key method for evaluating students' ability to read, understand, and reason through written programs as part of their computational development. However, in the current landscape, where students can obtain implemented solutions within seconds through appropriate prompts in ChatGPT (and related GenAI tools), this introduces a potential concern about reduced engagement in hands-on coding practices. This shift also underscores the need for new or adapted comprehension methods to ensure that students continue to develop the computational skills necessary for achieving programming proficiency.

2.2 Code Reviews

One notable comprehension model identified in the literature is the use of pedagogical code reviews (PCRs), which are structured peer review sessions aimed at deepening students' understanding of code. A central focus of PCRs is to expose students to essential coding practices, including code correctness, readability, and functionality. This type of inspection

enables students to recognize poor coding behaviors and habits, with the intent to correct such tendencies in efforts to employ stronger coding practices to promote efficiency and quality as they code [16]. Literature also highlights additional skills that students develop through the integration of PCRs into early computing courses, including communication, teamwork, peer-to-peer engagement, and interpersonal skills [17, 8]. Such skills are also essential for students to demonstrate as they seek career opportunities post graduation.

2.3 Verbal Assessments

Verbal assessments, also known as think-aloud assessments, offer another approach to evaluating students' comprehensive understanding. This model involves assessing a student's ability to articulate their thought process while solving a problem or completing a task [18]. These assessments are particularly effective in revealing students' reasoning strategies and metacognitive awareness. Ericsson and Simon described think-aloud protocols as a window into short-term memory (STM), emphasizing that only information actively attended to and processed in STM can be verbalized during task performance [19].

When emphasizing verbal assessments in education, Cowan advocates for the use of concurrent think-aloud protocols in educational action research. He argues that these methods provide immediate insight into students' performance and cognitive understanding, enabling instructors to better interpret the short-term cognitive operations that influence learning experiences. Furthermore, Cowan notes that thinking aloud reveals not only whether students rely on step-by-step reasoning or pattern-matching strategies, but also how they cognitively engage with a task [20].

In computing education literature, think-aloud assessments have been shown to be effective in evaluating students' ability to articulate their computational understanding. Vihavainen, Miller, and Settle [21], for instance, note that think-aloud assessments expose novice programmers to key practices such as decomposition, debugging, and conceptual clarification. These assessments make the learning process more observable to the instructor while facilitating a shared exchange of knowledge and understanding between student and instructor. Whalley et al. [22] found that think-aloud verbalization can reveal challenges to effective debugging practices, including imprecision, a lack of systematic strategies, and incomplete comprehension assessments.

Literature has increasingly examined verbal explanation–based assessments as a means of addressing unauthorized uses of generative AI, reflecting growing concerns about the reliability of written work for accurately verifying student understanding [23]. Prior work argues that traditional oral examinations must be reimagined for undergraduate education, emphasizing that the unplanned, real-time nature of a viva voce makes it more difficult for students to merely give the appearance of understanding compared to traditional written assessments [23]. Within computing and programming education specifically, researchers have begun exploring the integration of AI alongside oral and verbal explanation–based practices to better gauge student learning and promote intellectual accountability [24, 25, 26, 27, 28, 29, 30].

2.4 Preserving Academic Integrity in the Era of Generative AI

As GenAI tools continue to emerge and evolve, their increasing accessibility to students presents new challenges for educators, who must now navigate how to maintain and safeguard the integrity of learning in the classroom. Literature addressing these concerns is rapidly expanding. In higher education in particular, there is growing concern that GenAI tools may contribute to academic dishonesty and cheating. Kumar et al. [31], for instance, argue that GenAI-generated content is being misused to complete assignments and exams, often evading traditional plagiarism detection systems due to the sophistication of these tools. Rane et al. [32] highlight current interventions that universities are implementing to curb AI driven academic misconduct. One such strategy involves redesigning course assessments to emphasize problem-solving, creativity, and verbal examinations [32].

In computing education, efforts to protect student learning in the age of GenAI have also received growing attention. The literature frequently highlights ChatGPT as a prominent GenAI tool used in classroom-based computational learning settings [33, 34, 35]. In the context of learning to program, ChatGPT can instantly generate syntactically correct code on demand. This presents a challenge for educators, especially in determining whether students truly understand the underlying coding concepts or are merely relying on AI-generated solutions.

Thus, the use of comprehensive strategies such as pedagogical code reviews and verbal (think-aloud) assessments can serve as critical pedagogical interventions. These approaches provide instructors with deeper insight into students' authentic understanding and help safeguard against superficial engagement with GenAI output. As students verbalize their reasoning, educators can more effectively evaluate whether learners truly comprehend the logic behind their coding solutions, rather than merely reproducing or adopting responses generated by AI tools.

2.5 Conceptual Framework

This work draws upon the well-established and widely recognized framework of Bloom's Taxonomy. In the context of curriculum development, the articulation of established learning objectives, and pedagogical assessment, Bloom's Taxonomy of Educational Objectives has been identified as a commonly adopted framework among educators [36, 37]. When emphasizing programming pedagogy and computational learning, Bloom's Taxonomy allows for learning objectives that foster the ability to assess cognitive learning as students engage with mastering computing protocols and concepts [38]. When examining the six objectives that comprise Bloom's Taxonomy in its revised form, *remember*, *understand*, *apply*, *analyze*, *evaluate*, and *create* [39], each objective can be viewed as an essential element that supports success in programming pedagogy and computational learning. In introductory programming courses, students are expected to *remember* and *understand* fundamental concepts that can then be *applied* in the development of computational solutions. Protocols such as PCRs can be employed to assess students' ability to *analyze* and *evaluate* developed solutions in terms of correctness, readability, and functionality. Finally, students' capacity to *create* computational solutions serves as a clear indicator of the effectiveness of their learning and the extent to which knowledge has been acquired and developed.

Yet, as students gain increasing access to GenAI tools, it is essential that educators recognize this shift and make appropriate provisions to account for these tools during the learning experience. Simply prohibiting the use of GenAI in the classroom may be insufficient, as students may still access such technologies independently. As GenAI continues to evolve in sophistication and availability, educators must adapt their teaching and assessment strategies to maintain effective learning objectives and outcomes. Given the rapid advancement of artificial intelligence in computing, it is becoming more common for students to engage with these tools as they build their computational skills [33, 34, 35]. Therefore, it is important for assessment methods to move beyond a direct reliance on written deliverables and instead evaluate deeper indicators of student understanding and performance. One approach to assess such understanding and performance in depth are verbal assessments. As discussed in the Literature Review section, verbal assessments offer direct insight into students' thought processes [18], active short-term memory, and how they process and apply acquired information [19]. Moreover, they reveal students' cognitive reasoning when working with new material [20]. Verbal assessments also reduce the risk of plagiarism and address concerns regarding the authenticity of knowledge that may not be genuinely acquired.

Therefore, this work not only draws upon Bloom's Taxonomy as its foundational framework but also emphasizes the importance of capturing the authenticity of students' computational knowledge as it is acquired throughout the learning process, particularly in the presence of GenAI tools. Figure 1 illustrates potential differences between written and verbal explanations of acquired knowledge, highlighting issues of authenticity within the context of developing computational skills. A key distinction between these two forms of knowledge reporting lies in the perceived authenticity of student understanding. While written explanations may introduce uncertainty regarding the true extent of a student's computational knowledge, verbal assessments provide a clearer and more reliable window into authentic understanding and the ability to demonstrate skills. This framework further illustrates how students engage with a range of resources, from traditional instructional methods to AI-based tools, as part of their computational learning process. As students acquire knowledge, it is essential to assess not only the amount of content retained but also the authenticity of that knowledge. In this regard, written explanations may leave room for doubt in demonstrating firsthand understanding, whereas verbal explanations offer a more robust means of gauging the depth and authenticity of student learning.

3. Methods

This initiative implemented two targeted interventions to address the potential impact of the GenAI tool, ChatGPT, in an introductory programming course: code reviews and verbal explanations. The objective of this study was to examine students' ability to effectively use ChatGPT while completing a set of take-home assignments, involving the application of code review and/or verbal explanation practices to validate their understanding of targeted course topics. A mixed-methods approach was used to gather data on students' performance with incorporating ChatGPT on selected assignments, drawing on quantitative performance scores and qualitative comments and feedback provided by the grader. The qualitative comments and feedback provided by the grader were subjected to thematic analysis [40] to identify patterns and themes, highlighting the specific challenges and issues that prevented the students from attaining

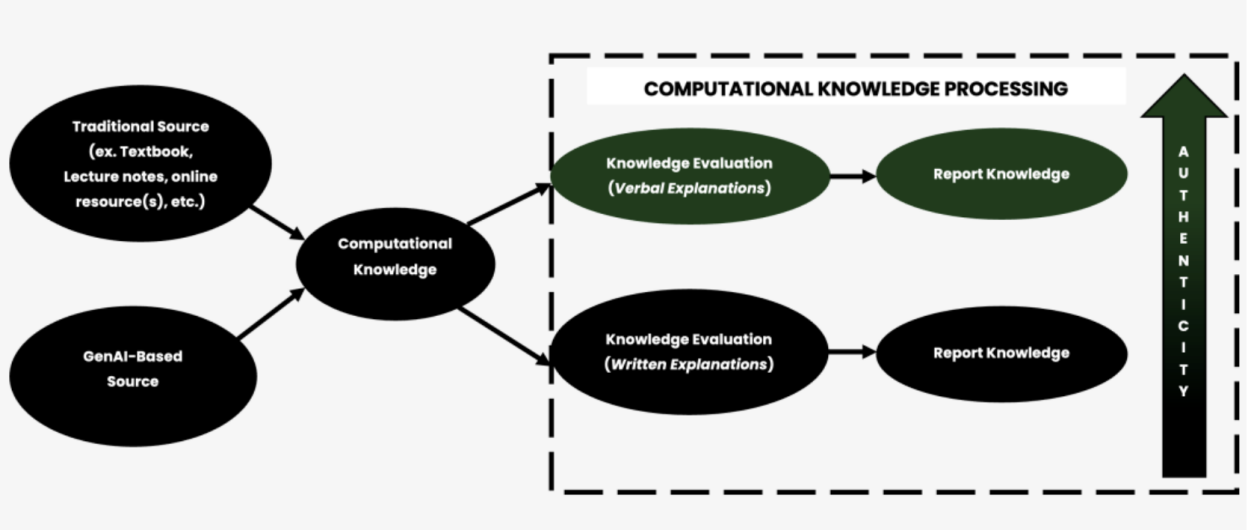

Figure 1: Computational Knowledge Processing & Authenticity: Written vs. Verbal Explanations

a perfect score on their deliverables. In addition, pre and post-surveys were administered to the students to collect quantitative responses reflecting their experiences with ChatGPT, specifically regarding its perceived impact to enhance their abilities to learn programming, critical thinking skills, and code review and evaluation skills. These surveys were part of a broader assessment administered across multiple sections of this introductory Java programming course. Aspects of this assessment have already been disseminated for publication [41]. The case study represented in this article reflects student performance from only one section of the course while explicitly examining their performance with using ChatGPT hands-on. Additionally, the survey data presented in this article reflect a small subset of questions administered to this targeted section to exclusively examine such hands-on experiences.

For the assignments and project in this study, students submitted their solutions as .java files, provided written analyses and reviews of their own or ChatGPT-generated Java solutions as .doc, .txt, or .pdf files, and/or recorded verbal analyses and reviews in audio or video formats (e.g., .mp3, .mp4, .m4a, .mov, or linked URLs) via Blackboard. For each assignment a corresponding grading rubric was assigned to evaluate the students' performance. The recorded sessions served as one of the criterion on the grading rubrics for the third take-home assignment and semester project.

3.1 Course Context & Design

The course spanned 17 weeks and enrolled 32 students, including 18 freshmen, 11 sophomores, and 3 juniors. As part of the course requirements, students completed three take-home programming assignments and one cumulative semester project, all aligned with the course objectives. Due to the timing of this study's implementation, the first take-home assignment did not include any code review or verbal explanation interventions. The incorporation of ChatGPT began with the second take-home assignment, which required the students to strictly use this AI tool to generate their programming assignment, and conduct a written code analysis/review on the

Table 1: Descriptive Statistics - Student Performance

Assignment Deliverable	**Targeted Course Topics**	**Imposed Interventions**
Take-Home Assignment #1	Elementary Programming Selections Mathematical Functions, Characters, and Strings	None
Take-Home Assignment #2	Conditions Loops Methods	Code Reviewing
Take-Home Assignment #3	Single-Dimensional Arrays Double-Dimensional Arrays Classes and Objects	Code Reviewing Verbal Explanation
Semester Project	All Topics (Selected Set)	Code Reviewing Verbal Explanation

Table 2: Grading Description for Second Take-Home Assignment

Criteria	Points Distribution Ranges
Initial Conceptualization	0 - 25 points
Generated Java Solution	0 - 25 points
Code Analysis & Correctness	0 - 25 points
Comparative Examination	0 - 25 points
TOTAL POINTS POSSIBLE	**100 points**

generated solution. More details regarding the protocol used for the ChatGPT-based assignments and project are discussed in further detail in the next set of paragraphs. Table 1 outlines the administered take-home assignments, the targeted topics, and the associated interventions.

For the second take-home assignment, students were first required to carefully review the problem description and associated Java implementation requirements. Rather than immediately coding a solution, they began by conceptualizing an ideal approach to identify relevant concepts, paradigms, and data structures needed to solve the problem. Next, students input the Java implementation requirements into ChatGPT to generate a corresponding solution. They then reviewed and analyzed the generated code to assess its correctness and understand its functional behavior. Finally, students completed a reflective assessment in which they compared their initial conceptualization with the ChatGPT-generated solution, noting both similarities and discrepancies between the two. Students who did not earn a perfect score on this assignment received feedback in Blackboard explaining the reasons for the point deductions. Table 2 presents descriptive details of the criteria used to grade the second take-home assignment. The key concepts and data structures highlighted in this assignment were *conditions*, *loops*, and *methods*.

For the third take-home assignment, students were instructed to carefully review the problem description, task overview, and code requirements. Rather than immediately code a solution, they first conceptualized the problem to identify relevant concepts, paradigms, and data structures needed for an effective solution. They then were required to develop pseudocode in plain text to outline their proposed solution. Using a Java editor of their choice, students then implemented their solution in Java. Afterward, they submitted the task overview and code requirements to

Table 3: Grading Description for Third Take-Home Assignment

Criteria	**Points Distribution Ranges**
Written Conceptualization	0 - 10 points
Written Pseudocode	0 - 10 points
Implemented Java Solution	0 - 30 points
Generated Java Solution	0 - 15 points
Video Recording of Code Examination & Analysis	0 - 10 points
Comparative Examination Between Both Solutions	0 - 25 points
TOTAL POINTS POSSIBLE	**100 points**

Table 4: Grading Description for Semester Project

Criteria	**Points Distribution Ranges**	**Final Score Distribution** (%)
Project Report	0 - 100 points	30%
Recorded Explanation (line-by-line)	0 - 100 points	30%
Implemented Solution	0 - 100 points	40%
Bonus: Implemented OOP concepts*	0 - 25 points*	<=25pts*
TOTAL POINTS POSSIBLE		**100; 125 (with bonus)***

ChatGPT to generate an alternative Java solution. Students then performed a recorded verbal review and analysis of both their own implementation and the ChatGPT-generated solution, evaluating each for code correctness and functional behavior. Finally, they completed a reflective assessment, comparing the two solutions and documenting key similarities and differences. Students who did not earn a perfect score on this assignment received feedback in Blackboard explaining the reasons for the point deductions. Table 3 presents descriptive details of the criteria used to grade the third take-home assignment. The key concepts and data structures highlighted in this assignment were *arrays*, *classes*, and *objects*.

The semester project was designed to give students an opportunity to apply their acquired knowledge creatively and independently by developing an original solution to a problem of their choice, based on topics (concepts and data structures) covered throughout the 17-week course. To promote originality, each student was required to submit a proposal outlining their project idea and intended approach. Additionally, they were provided with guidelines specifying the concepts, paradigms, and data structures that needed to be incorporated into their solution. For the final submission, students submitted their implemented Java program along with a recorded verbal review and analysis of their written code. This recording also included a live demonstration of their solution running in a Java editor of their choice. Students who did not earn a perfect score on this assignment received feedback in Blackboard explaining the reasons for the point deductions. Table 4 presents descriptive details of the criteria used to grade the semester project.

4. Results

Through this initiative, there were some notable outcomes reflecting the students' performance with programming in the presence of ChatGPT. Table 5 provides descriptive details regarding the students' scores on these three assignments (reflecting grading descriptions used as noted Tables

Table 5: Descriptive Statistics - Student Performance

Assignments	Take-Home Assignment 2	Take-Home Assignment 3	Semester Project
Number of Submissions	32	32	32
Possible Score	100	100	125
Highest Score Achieved	100	100	125
Mean Score	84.78	82.50	109.83
Median Score	97.50	97.50	115
Standard Deviation	30.41	30.67	23.50
# of Students with Possible Score	16	16	14*
# of Students who Received Point Deductions	16	16	18**

**denotes possible score (with or without bonus points)*
***denotes students who lost points, regardless of whether they attempted the bonus opportunity*

2-4).

For Take-Home Assignment 2, students achieved an average score of 84.78, with half of the class earning a perfect score. For those who did not receive a perfect score, the thematic analysis revealed the following themes reflecting specific challenges and issues that these students experienced: Code Analysis & Correctness, Comparative Examination, and Incomplete Submission. The Code Analysis & Correctness theme captured cases in which students provided an ChatGPT generated code, demonstrating limited understanding of the code's syntactical correctness, readability, and functional behavior. The Comparative Examination theme reflected difficulties in articulating similarities and differences between the student's own conceptualization and the code produced by ChatGPT. The Incomplete Submission theme referred to instances where students failed to submit some or all required components of the assignment.

For Take-Home Assignment 3, students collectively achieved an average score of 82.50, with half of the class earning a perfect score. For those who did not receive a perfect score, the thematic analysis revealed the following themes reflecting specific challenges and issues that these students experienced: Comparative Examination, Incomplete Submission, Incomplete Submission (No Video Recording), and Written Conceptualization. The Comparative Examination theme reflected students' difficulties in articulating the similarities and differences between their own code implementation and the code generated by ChatGPT. The Incomplete Submission theme captured cases in which students failed to submit multiple required components of the assignment. The Incomplete Submission (No Video Recording) theme specifically highlighted students who did not submit the required video recording demonstrating their code analysis and comparison. Lastly, the Written Conceptualization theme encompassed instances where students demonstrated a limited or inaccurate understanding of the problem, including misconceptions related to key concepts, paradigms, or data structures necessary for an effective solution.

For the semester project, students achieved an average score of 109.83, with slightly more than one-third of the class earning a perfect score, regardless of whether they attempted the bonus portion. Additionally, 8 of the 32 students chose not to attempt the bonus portion of the project, and 2 of those 8 still earned a perfect score (100). The thematic analysis conducted on the semester project revealed only one theme: Implemented Solution. The Implemented Solution theme reflected instances where students demonstrated discrepancies between their intended solution and its actual implementation. Notably, no issues were identified in students' verbal explanations of their code; all students were able to clearly articulate the functionality of their

Table 6: ChatGPT Experience: Likert-Scaled Questions

Likert Questions	**N**	**Mean**	**SD**
Enhanced Program Learning	16	5.81	1.11
Critical Thinking Skill Development	16	4.88	1.54
Coding Review & Evaluation Skill Development	16	5.44	1.55
SD = standard deviation			

programs line by line. Given the open-ended nature of the semester project, where students had full autonomy in selecting the problem they wished to solve, this suggests the possibility of a stronger overall understanding of their code and an enhanced ability to comprehend and explain their program's functionality effectively.

For Take-Home Assignments 2 and 3, approximately half of the students demonstrated a proficient understanding of reading written code and verbally explaining its syntactical correctness and functional behavior. Over time, this understanding improved, with the majority of students exhibiting such comprehension in their semester projects, though some students still displayed discrepancies in the correctness of their code implementations.

4.1 Survey Results

One question on the pre-survey specifically asked students about their prior experience with ChatGPT. Of the 26 out of 32 students who completed the pre-survey, 96% reported having at least some prior experience with ChatGPT. On the post-survey, three key questions (SQ) were included to assess how students used ChatGPT, particularly in the context of their in-course learning:

- SQ1: How much has ChatGPT improved the students' ability to learn how to program?
- SQ2: How much do the students think that ChatGPT has helped them build their critical thinking skills?
- SQ3: How much do they think that ChatGPT has helped them build their coding review/evaluation skills?

These three questions were measured using a 7-point Likert scale (1 = "Absolutely Not", 7 = "Absolutely Yes"). Table 6 presents the results, including the means and standard deviations (SD), to illustrate how students perceived their experiences with ChatGPT while learning to program.

Among the 16 students who completed this portion of the survey, results indicated moderately favorable perceptions from the students regarding their experiences with ChatGPT. As shown in Table 6, students reported a mean score of **5.81/7.00**, indicating a belief that ChatGPT supported the students' ability to learn programming. They also gave a mean score of **4.88/7.00**, reflecting a moderate belief that Chat-GPT contributed to the development of their critical thinking skills. Additionally, a mean score of **5.44/7.00** suggested that students found ChatGPT helpful in reviewing and evaluating code.

Additional statistical analyses were conducted on the three Likert scale questions to determine whether students gave significantly higher ratings to certain Likert questions over others. A one-way ANOVA was performed to examine differences among the mean scores for these questions; however, the results did not reveal any statistically significant differences.

5. Discussion

This work contributes meaningfully to support introductory programming and navigate the impact of generative AI in computing and engineering education. By integrating code reviews and verbal explanation assignments, this case study promotes conceptual understanding, academic integrity, and authentic assessment. The intervention presented here offer a practical, evidence based strategy that can be adapted across the computing education community, particularly within computational learning and programming pedagogy.

By requiring students to explain their code and analyze alternative solutions, the interventions promote the development of computational literacy, the capacity not only to write code but also to reason about it, compare approaches, and adapt solutions across contexts. This aligns with ongoing efforts in programming pedagogy that extend beyond code correctness to foster comprehension, design reasoning, and abstraction skills. The structured comparisons between student-developed code solutions and ChatGPT-generated code solutions prompted learners to examine discrepancies, justify decisions, and identify trade-offs, practices at the heart of critical thinking. These reflection tasks encouraged students to move beyond "what works" and consider "why it works," reinforcing their problem-solving frameworks and promoting principled decision-making in code development.

The availability of ChatGPT introduces complex pedagogical challenges: students can now submit functional code with little understanding of the underlying logic. This case study recognizes this reality and, rather than ignoring or prohibiting it, proposes a constructive and proactive response. By embedding ChatGPT into assignments as a complementary tool rather than a substitute for student effort, this approach reinforces student reasoning as the core of assessment. The design of verbal explanation tasks and coding reviews serve as a pedagogical countermeasure to GenAI over-reliance, ensuring that students can demonstrate genuine learning even in the age of generative AI. These findings and practices are highly relevant to computing educators who are navigating both the opportunities and challenges that AI presents in computing education.

This case study also introduces a novel instructional response to the pedagogical challenges posed by generative AI tools in programming education. The central innovation lies in the requirement that students explain their code and reasoning aloud, thereby transforming comprehension from an invisible process into a measurable performance. Instructors are no longer limited to assessing correctness alone; they can now evaluate understanding, clarity, logic, and reasoning through students' verbal walkthroughs. This shift from product-oriented assessment to process-based validation is especially important in GenAI contexts, where working code can be easily obtained without cognitive engagement. Verbal explanations serve as a proxy for comprehension. Students cannot fake understanding in a recorded walkthrough. The ability to articulate how a piece of code works, why certain structures were used, or how their implementation compares to an

AI-generated alternative, offers instructors a robust, scalable method for ensuring that students are learning meaningfully.

6. Limitations

This case study has noteworthy limitations that may pose potential threats to its validity. First, the study involved only 32 students from a single institution. This limitation highlights the need to examine the impact of coding reviews and verbal explanations at a larger scale, including multiple course sections, courses with higher student enrollments, and potentially multiple institutions, to better support the validity of the findings. Expanding the scope of this work would also address another limitation by enabling the inclusion of a control group, which could establish a baseline or allow for systematic comparison with traditional instructional approaches. Additionally, survey responses were obtained from only 50% of the enrolled students (N = 16). Collecting survey data from a larger proportion of participants would further strengthen the validity and robustness of the study's conclusions.

7. Future Work

The outcomes of this intervention highlight promising directions for future work aimed at supporting computational learning in the age of GenAI. Building on the current findings, the first author has implemented additional interventions in two programming courses during the Fall 2025 semester: an introductory information systems course focused on logic and structured design with Python, and the sequel to the introductory Java course featured in this report. The goal is to further explore how intervention models can promote student learning by assessing their understanding of core computational concepts. It is also planned to refine the current strategies, such as video-based verbal explanations and coding review assignments by incorporating lower-stakes assessments that integrate both methods. This adjustment aims to increase student familiarity, accessibility, and comfort with these approaches, making them more effective and sustainable as part of regular coursework. Aforementioned in the Limitations section, including additional courses will also increase the pool of participants with hope of strengthening the validity of this work. Another future work will be to partner with introductory programming courses at other institutions to study the impact of this approach on a larger scale with the intent to further strengthen its validity.

8. Conclusion

The emergence of GenAI tools has become increasingly prevalent and accessible for students as they learn. In the context of computing education and AI in education, this accessibility presents challenges both for students who are expected to develop computational knowledge and programming proficiency, and instructors who must design and administer assessments that accurately gauge the students' learning and understanding. This intervention was designed to preserve meaningful computational development while intentionally integrating a GenAI tool like ChatGPT into the learning process. This approach demonstrated potential in helping students articulate and showcase their code comprehension skills, even while having direct access to a

GenAI tool during their learning process. Moreover, the post survey revealed positive feedback on the potential benefits of GenAI tools like ChatGPT in supporting students as they learn to program and develop computational thinking skills. While these findings are promising, further work is needed to refine and expand this model, with the broader goal of sustaining effective computational learning and programming pedagogy in the age of generative AI.

References

[1] B. Adelson, "When novices surpass experts: The difficulty of a task may increase with expertise," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 10, no. 3, pp. 483–495, 1984, reprinted in Human Factors in Software Development (2nd ed.); Bill Curtis (ed.), IEEE Computer Society Press, 1985, pp. 55–67.

[2] M. Behroozi, C. Parnin, and T. Barik, "Hiring is broken: What do developers say about technical interviews?" pp. 1–9, 2019.

[3] D. Ford, T. Barik, L. Rand-Pickett, and C. Parnin, "The techtalk balance: what technical interviewers expect from technical candidates," in *Proceedings of the 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2017, pp. 43–48.

[4] N. Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognitive Psychology*, vol. 19, no. 3, pp. 295–341, 1987.

[5] V. Ramalingam and S. Wiedenbeck, "An empirical study of novice program comprehension in the imperative and object-oriented styles," in *Papers presented at the seventh workshop on Empirical studies of programmers*, 1997, pp. 124–139.

[6] L. Gugerty and G. M. Olson, "Comprehension differences in debugging by skilled and novice programmers," in *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, 1986, pp. 13–27.

[7] C. Izu, C. Schulte, A. Aggarwal, Q. Cutts, R. Duran, M. Gutica, B. Heinemann, E. Kraemer, V. Lonati, C. Mirolo *et al.*, "Fostering program comprehension in novice programmers-learning activities and learning trajectories," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, 2019, pp. 27–52.

[8] C. D. Hundhausen, A. Agrawal, and P. Agarwal, "Talking about code: Integrating pedagogical code reviews into early computing courses," *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 3, pp. 14:1–14:28, 2013.

[9] S. Oeda and H. Kosaku, "Development of a check sheet for code-review towards improvement of skill level of novice programmers," *Procedia Computer Science*, vol. 126, pp. 841–849, 2018.

[10] R. Bednarik and M. Tukiainen, "Effects of display blurring on the behavior of novices and experts during program debugging," in *CHI'05 Extended abstracts on human factors in computing systems*, 2005, pp. 1204–1207.

[11] P. Romero, R. Cox, B. du Boulay, and R. Lutz, "Visual attention and representation switching during java program debugging: A study using the restricted focus viewer," in *Diagrammatic Representation and Inference: Second International Conference, Diagrams 2002 Callaway Gardens, GA, USA, April 18–20, 2002 Proceedings 2*. Springer, 2002, pp. 221–235.

[12] S. Yusuf, H. Kagdi, and J. I. Maletic, "Assessing the comprehension of uml class diagrams via eye tracking," in *Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC)*. IEEE Computer Society, 2007, pp. 113–122.

[13] T. Kelly and J. Buckley, "A context-aware analysis scheme for bloom's taxonomy," in *14th IEEE International Conference on Program Comprehension (ICPC'06)*. IEEE, 2006, pp. 275–284.

[14] D. A. McMeekin, B. R. von Konsky, E. Chang, and D. J. Cooper, "Checklist inspections and modifications: applying bloom's taxonomy to categorise developer comprehension," in *2008 16th IEEE International Conference on Program Comprehension*. IEEE, 2008, pp. 224–229.

[15] L. E. Winslow, "Programming pedagogy—a psychological overview," *SIGCSE Bulletin*, vol. 28, no. 3, pp. 17–22, 1996.

[16] Y. Lu, T. Mao, X.and Wang, G. Yin, and Z. Li, "Improving students' programming quality with the continuous inspection process: a social coding perspective," *Frontiers of Computer Science*, vol. 14, pp. 1–18, 2020.

[17] T. Brown, M. R. Narasareddygari, M. Singh, and G. Walia, "Using peer code review to support pedagogy in an introductory computer programming course," in *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1–7.

[18] T. VanDeGrift, "Giving voice to problem-solving: Hearing students' techniques in video reflections," in *2024 ASEE Annual Conference & Exposition*, 2024.

[19] K. A. Ericsson and H. A. Simon, "Protocol analysis: Verbal reports as data," 1993.

[20] J. Cowan, "The potential of cognitive think-aloud protocols for educational action research," *Active Learning in Higher Education*, vol. 20, no. 3, pp. 219–232, 2019.

[21] A. Vihavainen, C. S. Miller, and A. Settle, "Benefits of self-explanation in introductory programming," in *Proceedings of the 46th ACM technical symposium on computer science education*, 2015, pp. 284–289.

[22] J. Whalley, A. Settle, and A. Luxton-Reilly, "A think-aloud study of novice debugging," *ACM Transactions on Computing Education (TOCE)*, vol. 23, no. 2, pp. 28:1–28:38, June 2023.

[23] P. Eachempati, R. Komattil, and A. Arakala, "Should oral examination be reimagined in the era of ai?" *Advances in Physiology Education*, vol. 49, no. 1, p. 208–209, Mar. 2025. [Online]. Available: http://dx.doi.org/10.1152/advan.00191.2024

[24] J.-T. Hung, C. Cui, D. M. Popescu, S. Chatterjee, and T. Starner, "Socratic mind: Scalable oral assessment powered by ai," in *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, ser. L@S '24. ACM, Jul. 2024, p. 340–345. [Online]. Available: http://dx.doi.org/10.1145/3657604.3664661

[25] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in *Proceedings of the 2022 ACM conference on international computing education research-volume 1*, 2022, pp. 27–43.

[26] P. Ohmann and E. Novak, "A multi-institutional assessment of oral exams in software courses," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 2025, pp. 882–888.

[27] S. J. Reckinger, "Asynchronous oral proficiency exams in a cs1 course," in *Proceedings of the ACM Global on Computing Education Conference 2025 Vol 2*, 2025, pp. 409–410.

[28] S. Kannam, Y. Yang, A. Dharm, and K. Lin, "Code interviews: Design and evaluation of a more authentic assessment for introductory programming assignments," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 2025, pp. 554–560.

[29] R. Azoulay, T. Hirst, and S. Reches, "Large language models in computer science classrooms: Ethical challenges and strategic solutions." *Applied Sciences (2076-3417)*, vol. 15, no. 4, 2025.

[30] S. Hooshangi, A. Shakil, S. Riddle, I. Aydin, N. Nasir, T. Parupudi, A. Rehman, M. Scott, J. Vahrenhold, A. Weerasinghe *et al.*, "Evaluating assessment practices in team-based computing capstone projects," *Proceedings of the 2025 Working Group Reports on Innovation and Technology in Computer Science Education*, 2026.

[31] A. Kumar, A. Kumar, S. Bhoyar, and A. K. Mishra, "Does chatgpt foster academic misconduct in the future?" *Public Administration and Policy*, vol. 27, no. 2, pp. 140–153, 2024.

[32] N. Rane, S. Shirke, S. P. Choudhary, and J. Rane, "Artificial intelligence in education: A swot analysis of chatgpt and its impact on academic integrity and research," *Journal of ELT Studies*, vol. 1, no. 1, pp. 16–35, 2024.

[33] R. Budhiraja, I. Joshi, J. S. Challa, H. D. Akolekar, and D. Kumar, ""it's not like jarvis, but it's pretty close!"-examining chatgpt's usage among undergraduate students in computer science," in *Proceedings of the 26th Australasian Computing Education Conference*, 2024, pp. 124–133.

[34] L. Kloub and A. Gupta, "Chatgpt in computer science education: exploring benefits, challenges, and ethical considerations," in *ASEE North East section*, 2024.

[35] Y. Xue, H. Chen, G. R. Bai, R. Tairas, and Y. Huang, "Does chatgpt help with introductory programming? an experiment of students using chatgpt in cs1," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, 2024, pp. 331–341.

[36] T. V. Ramirez, "On pedagogy of personality assessment: application of bloom's taxonomy of educational objectives," *Journal of personality assessment*, vol. 99, no. 2, pp. 146–152, 2017.

[37] C. P. Ormell, "Bloom's taxonomy and the objectives of education," *Educational Research*, vol. 17, no. 1, pp. 3–18, 1974.

[38] P. Mahatanankoon and J. Wolf, "Cognitive learning strategies in an introductory computer programming course." *Information Systems Education Journal*, vol. 19, no. 3, pp. 11–20, 2021.

[39] D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," *Theory into practice*, vol. 41, no. 4, pp. 212–218, 2002.

[40] V. Braun and V. Clarke, *Thematic analysis.* American Psychological Association, 2012.

[41] E. Dillon, P. Ordóñez, U. Hasan, O. N. Owoeye, and S. R. Dharmale, "Exploring the impact of chatgpt on early information systems majors: Opportunities and challenges in learning to program," in *2025 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2025, pp. 1–9.