Detecting and Defining Responsible Use of Generative AI in Introductory Computer Science

Sebastian Vargas, James Hinojos, Jesus Ramirez, and Christian Servin

Abstract The use of Generative Artificial Intelligence (GenAI) in education has increased rapidly with the adoption of tools such as ChatGPT, Gemini, Copilot, and Claude. These systems provide students with immediate access to explanations, code generation, debugging suggestions, and example solutions, fundamentally changing how learners approach programming tasks. While these technologies create new opportunities for learning support and productivity, their integration has raised concerns regarding academic integrity and assessment practices. Computing educators have long developed expertise in recognizing patterns in student work. Through experience, instructors often develop an intuitive *sense of degree* when evaluating programming assignments, allowing them to distinguish between authentic student effort and work heavily assisted by automated tools. This study reports findings from a two-year college investigation examining programming assignments in introductory computing courses to identify segments likely assisted by AI tools. We analyze patterns that suggest GenAI involvement and categorize the types of assistance students obtain from these systems. Additionally, we explore students' reported motivations for using these tools, including concept clarification, debugging support, and access to additional examples. Understanding these behavioral patterns can contribute to responsible and pedagogically aligned integration of AI in computing education.

Sebastian Vargas Rivera and Jesus Ramirez
University of Texas at El Paso, 500 W. University
El Paso, TX 79968, USA,
e-mail: svargasrivera@miners.utep.edu/jmramirez32@miners.utep.edu

James Hinojos and Christian Servin
ITS/CS, El Paso Community College, 919 Hunter Dr.
El Paso, TX 79915, USA,
e-mail: jhino121@my.epcc.edu/cservin1@epcc.edu

1 Introduction

Computer Science courses—particularly the traditional CS0 (Introduction to Computing), CS1 (Introduction to Programming), and Data Structures—are widely recognized as the intellectual foundation of the computing discipline. These courses establish the conceptual, procedural, and analytical principles that shape the first two years of computer science education, e.g., see: [3]. They introduce students to algorithmic thinking, abstraction, problem decomposition, modular design, and the systematic development of software artifacts. More importantly, they cultivate the habits of mind that enable learners to transition from novice programmers to disciplined computing professionals.

In the United States, these foundational courses play an especially critical role in two-year institutions—commonly known as community and technical colleges, i.e., [2, 1]. Such programs serve as primary entry points into computing for a diverse population of learners, including first-generation college students, working adults, military veterans, and career changers. Two-year programs are central to effective *2+2 articulation pathways*, allowing students to begin their studies in smaller, more accessible institutions, complete their general education core and lower-division computing requirements, and then transfer seamlessly into four-year universities or institutes to complete bachelor's degrees. The integrity and alignment of CS0, CS1, and Data Structures within these pathways are therefore essential to ensuring academic mobility, workforce readiness, and long-term student success.

Because of this transfer-oriented mission, foundational computing courses must be carefully designed, clearly articulated, and consistently proctored. Curriculum misalignment, ambiguous assessment practices, or uneven rigor can create downstream challenges for students entering upper-division coursework. Well-defined learning outcomes, structured programming assignments, and transparent evaluation criteria are necessary to guarantee that students acquire not only syntactic familiarity with programming languages but also deep conceptual understanding and transferable problem-solving skills.

2 GenAI Programming Paradigm: The Need for an Emerging Programming Model in Computing

The rapid emergence of Artificial Intelligence (AI)—particularly Generative AI (GenAI) tools—has introduced new dynamics into foundational computing education. Tools capable of generating code, explanations, and debugging suggestions are increasingly accessible to students in introductory programming courses. While these systems offer opportunities for scaffolding learning, accelerating feedback, and promoting experimentation, they also raise important pedagogical questions. Specifically, there is a growing need to examine *how* these tools are being used

in computer programming courses and to identify the "whats" and the "hows" of student interaction with AI-assisted development, e.g., see:[4]

Involving students in structured research about their own AI usage can illuminate patterns of engagement, dependency, misunderstanding, and strategic learning. For instance, a student may clearly understand the overall objective of an assignment or assessment, yet struggle with ambiguous instructions, missing technical details, or unclear constraints. In such cases, GenAI tools may be used not as substitutes for learning, but as clarifiers of expectations. Conversely, there are scenarios in which assignment prompts lack sufficient precision—whether due to oversight or incomplete specification—leading students to rely on AI systems to interpret missing requirements. In other situations, students may use AI to generate complete solutions without fully engaging with the underlying logic, creating a gap between perceived competence and actual mastery.

These emerging patterns underscore the importance of systematically studying AI integration within foundational computing courses. Rather than framing AI solely as a threat to academic integrity, there is value in investigating responsible and pedagogically aligned uses. Understanding where AI assists learning, where it obscures conceptual development, and where it compensates for instructional ambiguity can inform improved assignment design, clearer specifications, and more robust assessment strategies.

For two-year transfer institutions in particular, this inquiry is especially urgent. These programs must preserve rigor, ensure alignment with four-year expectations, and simultaneously support students navigating rapidly evolving technological landscapes. Engaging students as research participants—and in some cases as co-investigators—provides an evidence-based pathway to refine pedagogical practices while preparing learners to interact ethically and effectively with AI-driven tools.

Ultimately, the convergence of foundational computer science education and generative AI technologies calls for a balanced, research-informed approach. By examining how students use AI in CS0, CS1, and Data Structures, educators can better define responsible integration strategies that strengthen learning outcomes, maintain transfer integrity, and cultivate reflective, self-regulated computing professionals.

3 Constraint Programming Techniques in Computing Education

In this work, we propose an interface that allows students to interact with Generative AI (GenAI) tools while enabling the identification of the stages in which such tools are used during the programming process as shown in Figure 1. Students rely on GenAI for a variety of reasons, including concept clarification, debugging assistance, and code generation. Our approach tracks when and why these tools are used throughout the development workflow. This perspective is particularly relevant to constraint programming techniques, as the components of the proposed intelligent system are designed to detect and represent the stages where GenAI assistance

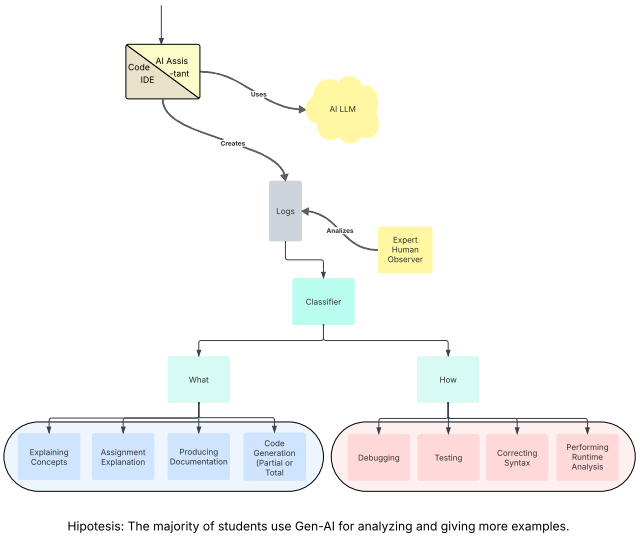

Fig. 1 Example of GenAI usage classification workflow.

occurs. Constraint-based approaches provide a declarative and efficient methodology for representing and solving many practical problems. However, although these techniques have been widely explored within the computing and programming communities, their application within educational contexts remains limited. This work explores how constraint-based representations can be leveraged to better understand and analyze student interactions with GenAI in programming environments.

References

1. C. Servin, E. K. Hawthorne, L. Postner, C. Tang, and C. S. Tucker, “Curricular and pedagogical considerations in computer science education: The role of community colleges for the next decade”, *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024)*, Portland, OR, USA, 2024, pp. 1196–1201. DOI: https://doi.org/10.1145/3626252.3630819.
2. C. Tang, “Community College Corner: Community colleges in the United States and around the world”, *ACM Inroads*, Vol. 8, No. 1, 2017, pp. 21–23.
3. C. Servin, M. Geissler, P. Schmelz, C. Tang, and C. Tucker, “CS Transfer guidelines for two-year programs: A curricular model integrating AI, cybersecurity, ethics, and the profession”, *Proceedings of the 57th ACM Technical Symposium on Computer Science Education (SIGCSE TS 2026)*, 2026, pp. 1507–1508. DOI: https://doi.org/10.1145/3770761.3777267.
4. A. Scholl and N. Kiesler, “How novice programmers use and experience ChatGPT when solving programming exercises in an introductory course”, *Proceedings of the 2024 IEEE Frontiers in Education Conference (FIE)*, Washington, D.C., USA, 2024, pp. 1–9. DOI: https://doi.org/10.1109/FIE61694.2024.10893442.