The Journal of Computing Sciences in Colleges

Papers of the 33th Annual CCSC Midwestern Conference

October 9th-10th, 2026
Kenyon College
Gambier, OH

Bin Peng, Associate Editor
Park University

William Turner, Regional Editor
Wabash College

Volume 42, Number 2 **October 2026**

The Journal of Computing Sciences in Colleges (ISSN 1937-4771 print, 1937-4763 digital) is published at least six times per year and constitutes the refereed papers of regional conferences sponsored by the Consortium for Computing Sciences in Colleges.

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

Following is a listing of the contact information for the members of the Board of Directors and the Officers of the Consortium for Computing Sciences in Colleges (along with the years of expiration of their terms), as well as members serving CCSC:

Shereen Khoja, President (2027), shereen@pacificu.edu, Computer Science, Pacific University, Forest Grove, OR 97116.

Michael Flinn, Vice President/President-Elect (2027), mflinn@frostburg.edu, Department of Computer Science & Information Technologies, Frostburg State University, Frostburg, MD 21532.

Abbas Attarwala, Publications Chair (2027), aattarwala@csuchico.edu, Department of Computer Science, California State University Chico, Chico, CA 95929.

Ed Lindoo, Treasurer (2029), elindoo@regis.edu, Anderson College of Business and Computing, Regis University, Denver, CO 80221.

Haiyan Cheng, Membership Secretary (2028), hcheng@willamette.edu, Department of Computer Science, Willamette University, Salem, OR 97301.

Brian Hare, Central Plains Representative (2029), hareb@umkc.edu, University of Missouri-Kansas City, Kansas City, MO 64110.

Pranshu Gupta, Eastern Representative (2029), Pranshu.Gupta@desales.edu, College of Sciences, DeSales University, Center Valley, PA 18034.

Gabe Ferrer, Midsouth Representative (2028), ferrer@hendrix.edu, Department of Computer Science, Hendrix College, Conway, AR 72032.

David Largent, Midwest Representative (2029), dllargent@bsu.edu, Department of Computer Science, Ball State University, Muncie, IN 47306.

Michael Gousie, Northeastern Representative (2028), gousie_mike@wheatoncollege.edu, Computer Science Department, Wheaton College, Norton, MA 02766.

Ben Tribelhorn, Northwestern Representative (2027), tribelhb@up.edu, School of Engineering, University of Portland, Portland, OR 97203.

Mohamed Lotfy, Rocky Mountain Representative (2028), MohamedL@uvu.edu, Information Systems & Technology Department, College of Engineering & Technology, Utah Valley University, Orem, UT 84058.

Mika Morgan, South Central Representative (2027), mika.morgan@msutexas.edu, Department of Computer Science, Midwestern State University, Wichita Falls, TX 76308.

Karen Works, Southeastern Representative (2027), keworks@pc.fsu.edu, Department of Computer Science, Florida State University Panama City, Panama City, FL 32405.

Michael Shindler, Southwestern Representative (2029), mikes@uci.edu, Computer Science Department, UC Irvine, Irvine, CA 92697.

Serving the CCSC: These members are serving in positions as indicated:
Bin Peng, Associate Editor, bin.peng@park.edu, Computer Science and Information Systems, Park University, Parkville, MO 64152.
Lucas Cordova, Associate Treasurer, lpcordova@willamette.edu, Department of Computer Science, Willamette University, Salem, OR 97301.
George Dimitoglou, Comptroller, dimitoglou@hood.edu, Department of Computer Science, Hood College, Frederick, MD 21701.
Megan Thomas, Membership System Administrator, mthomas@cs.csustan.edu, Department of Computer Science, California State University Stanislaus, Turlock, CA 95382.

Karina Assiter, National Partners Chair, KarinaAssiter@landmark.edu, Landmark College, Putney, VT 05346.

Ed Lindoo, UPE Liaison, elindoo@regis.edu, Anderson College of Business and Computing, Regis University, Denver, CO 80221.

Deborah Hwang, Webmaster, hwangdjh@acm.org.

CCSC National Partners

The Consortium is very happy to have the following as National Partners. If you have the opportunity please thank them for their support of computing in teaching institutions. As National Partners they are invited to participate in our regional conferences. Visit with their representatives there.

Platinum Level Partner

College Board

Gold Level Partner

ACM ACM2Y
CCECC
Blossoms
MAP-CS: Mission-Aligned Programs in Computer Science
Rephactor

Welcome to the 2026 CCSC Midwestern Conference

Welcome to the 33rd annual CCSC Midwest conference. We are all excited to have you here in lovely Gambier, Ohio. We are especially grateful to James Skon and everyone at Kenyon College for sharing their new state-of-the-art facilities.

Once again, this year's conference brings educators, students, industry professionals, and others together to learn about the latest innovations and challenges in computing. We are also looking forward to seeing how our students showcase their skill and creativity during the hackathon.

This year we have 12 strong papers chosen from 19 submissions (a 63% acceptance rate). The program also includes a tutorial, two workshops, a roundtable discussion, several lightning talks, Nifty Assignments, and posters. We had 35 colleagues across the region and country serve as reviewers, and we sincerely appreciate the generosity of their time and effort. We are especially honored to have Mark Guzdial and Barb Erickson as our featured speakers and look forward to their presentations. We also thank our National Partners College Board, ACM2Y & CCSCC, Rephactor, BlossomsAI, and MAP-CS for their continued support of our activities.

We have a full schedule of events and encourage you to make the most of the many opportunities for learning and networking. Your participation is critical to the success of this conference, and we hope you find the sessions engaging and inspiring. Thank you for joining us, and we look forward to a productive and enjoyable conference!

Zachary Kurmas
Grand Valley State University
Conference Chair

James Skon
Kenyon College
Host

Imad Al Saeed
Saint Xavier University
Papers Chair

2026 CCSC Midwestern Conference Steering Committee

Conference Chair
Zachary Kurmas ..... Grand Valley State University, MI
Vice Conference Chair
William Turner ..... Wabash College, IN
Past Conference Chair
Paul Talaga ..... University of Indianapolis, IN
Site Chair
James Skon ..... Kenyon College, OH
Papers
Imad Al Saeed ..... Saint Xavier University, IL
Nifty Tools & Assignments
Ahmed Elmagrous ..... University of Wisconsin-Stout, WI
(Assistant) Fola Olagbemi ..... Hope College, MI
Work In Progress (WIP)
Ahmed Elmagrous ..... University of Wisconsin-Stout, WI
Panels, Tutorials, & Workshops
Sugandha Malviya ..... Ball State University, IN
(Assistant) Cathy Bareiss ..... Bethel University, MN
Authors
William Turner ..... Wabash College, IN
Programming Contest
(chair) Paul Talaga ..... University of Indianapolis, IN
(Assistant) Md Haque ..... Butler University, IN
(Assistant) Nathan Sommer ..... Xavier University, OH
Publicity
David Largent ..... Ball State University, IN
Speakers Chair
Stefan Brandle ..... Taylor University, IN
Student Showcase Chair
Andy Harris ..... Ball State University, IN
Two-year College Liaison
Kris Roberts ..... Ivy Tech Community College, IN
Vendors
Takako Soma ..... Illinois College, IL
Virtual Host
Cathy Bareiss ..... Bethel University, IN
Assistant Webmaster
William Turner ..... Wabash College, IN

Regional Board — 2026 CCSC Midwestern Region

Regional Representative
David Largent (2029) ..... Ball State University, IN
Regional Editor
William Turner (2027) ..... Wabash College, IN
Registrar
Deborah Hwang (2028) ..... University of Evansville (Retired), IN
Treasurer
Dominic Wilson (2026) ..... University of Findlay, OH
Webmaster
VACANT
CCSC At-Large Board Member
Karl Schmitt (2026) ..... Trinity Christian College, IL
Jeff Lehman (2027) ..... Huntington University, IN

Getting Students to Think While Learning to Program in the Age of Generative AI*

Keynote Speech

Barbara Ericson
Associate Professor, University of Michigan

Abstract

Programming class instructors are wrestling with the impact of generative AI. Allowing students to use generative AI can widen the performance gap between strong and weak students. Weak students are more likely to have generative AI just do their work for them, without any cognitive effort on their part. Active, social, and/or creative assignments can encourage students to still think in the age of generative AI. Dr. Ericson has been exploring free and interactive ebooks, mixed-up code (Parsons) problems, Peer Instruction, Process Oriented Guided Inquiry Learning (POGIL), and open-ended and creative projects. She also helped redesign a CS1 and CS2 course to leverage AI for the Consortium for Generative AI in CS Education. In addition, her students have been creating new types of practice problems that attempt to harness the power of GenAI *and* improve learning.

Bio

Dr. Ericson got her PhD in Human Centered Computing in 2018 from Georgia Tech and is an Associate Professor at the University of Michigan. She applies research results from educational psychology to help students learn to

program. For over 10 years, she has been creating free and interactive ebooks with new types of practice problems on the free and open-source ebook platform Runestone Academy. She won the 2022 ACM SGICSE Award for Outstanding Contributions to Computer Science Education and is a distinguished member of the ACM.

Beyond just Coding: Computing for Discovery, Expression, and Democracy*

Banquet Speech

Mark Guzdial
Professor, University of Michigan

Abstract

Computer science was invented as a medium for discovery and expression, to be taught to everyone. The early computer scientists worried about the danger of having this powerful new technology controlled by only a few. When access to computing is limited to a few, societies limit innovation and suffer fragile democracy. When access is broad—and learning is contextualized—students across disciplines can use computing to investigate data, communicate ideas, and scrutinize the systems that shape their lives. This talk describes how to design computing education that is technically authentic, relevant across disciplines, and advances democratic values.

Bio

Mark Guzdial is a Professor in Computer Science & Engineering and Director of the Program in Computing for the Arts and Sciences at the University of Michigan. He studies how people come to understand computing and how to make that more effective. He was one of the founders of the International Computing Education Research conference. He was a lead on the NSF alliance "Expanding Computing Education Pathways" which helped US states improve and broaden their computing education. He invented and has written

several books on the "Media Computation" contextualized approach to computing education. With his wife and colleague, Barbara Ericson, he received the 2010 ACM Karl V. Karlstrom Outstanding Educator award. He is an ACM Distinguished Educator and a Fellow of the ACM. He has just completed the second edition of his book, Learner-Centered Design of Computing Education: Research on Computing for Everyone. He received the 2019 ACM SIGCSE Outstanding Contributions to Education award.

AI-Mediated Collaborative Cognition: Designing Systems That Preserve Reasoning in the Age of LLMs*

James Paul Skon[1,2]
[1]*Mathematics and Computer Science*
Kenyon College
Gambier, Ohio, USA
[2]*Karolinska Institutet*
Stockholm, Sweden
skonjp@kenyon.edu

Abstract

Large Language Models (LLMs) fundamentally weaken the relationship between student reasoning and observable output in computing education. When correct answers can be generated with minimal cognitive effort, traditional signals of understanding degrade, creating the risk of *cognitive bypass*—the production of correct results without engagement in the reasoning processes those results are intended to reflect. This paper argues that addressing this challenge requires a shift from tool-based integration of AI toward system-level design that enforces reasoning as a necessary condition for progress. Rather than treating AI as an open-ended assistant, we propose a constraint-based approach in which collaboration structure, interaction rules, and AI behavior are explicitly controlled to preserve cognitive engagement. We present coLearn-AI as an instance of this design approach. The system enforces role-based collaboration and turn-taking, constrains AI interaction to a scaffolding role, and maintains an append-only record of student interaction that

captures intermediate reasoning steps. It supports natural-language responses, executable Python and C++ code, and analysis of program output, allowing AI feedback to operate across common computing artifacts while being constrained toward guidance rather than direct solution generation. Preliminary classroom observations suggest more consistently distributed participation, less divide-and-conquer behavior, and greater instructor visibility into student reasoning processes. This work contributes a design perspective in which preserving reasoning requires not more capable AI, but carefully engineered constraints on how AI and students interact.

1 Introduction

The rapid adoption of Large Language Models (LLMs) in computing education has created a fundamental tension: while these systems can enhance productivity, they also decouple observable output from the cognitive effort traditionally required to produce it.

This shift undermines both assessment and learning. If output is no longer a reliable proxy for reasoning, educators must reconsider how learning environments are structured. The central question becomes:

How can we design systems where AI strengthens, rather than bypasses, the learning process?

Learning theory suggests that reasoning cannot be inferred solely from final outcomes, but must be observed through the processes that produce them. In environments where AI can generate correct outputs independently, preserving these processes becomes a central design challenge.

Prior work, including our own, has focused on building systems that integrate AI into guided inquiry workflows. In this paper, we shift focus from system features to the underlying design problem introduced by LLMs: the breakdown of the relationship between reasoning and observable output. We argue that addressing this requires not additional AI capability, but system-level constraints that preserve reasoning as a necessary condition for progress.

This paper proposes a design-oriented solution grounded in learning theory and illustrated through a collaborative platform, coLearn-AI. Rather than introducing new AI capabilities, the system demonstrates how structured interaction constraints can preserve reasoning as a necessary condition for progress in AI-mediated learning environments.

2 Related Work

2.1 AI in Education

Artificial Intelligence (AI) has been widely explored as a tool to support learning through intelligent tutoring systems, adaptive feedback, and automated assessment. Recent work on Large Language Models (LLMs) emphasizes personalized instruction and immediate feedback [3, 5].

However, much of this work focuses on *individual learning*, where AI functions as a tutor or answer generator. While effective for efficiency and access, this approach risks reducing opportunities for productive struggle and deep cognitive engagement.

2.2 AI and Collective Intelligence

Recent work has begun to examine how AI can support *collective intelligence* in educational settings. Casebourne et al. [1] argue that AI can facilitate group reasoning, shared knowledge construction, and collaborative problem solving.

This perspective aligns with social constructivist views of learning as an interaction among individuals, tools, and representations, positioning AI as a participant in a broader cognitive system rather than a replacement for human reasoning.

2.3 Positioning This Work

Prior work on AI-supported collaboration emphasizes enabling interaction, often assuming that effective collaboration will emerge when tools are provided.

In contrast, we argue that collaboration must be *structurally enforced*. Rather than treating AI as an open-ended assistant, we impose explicit constraints on interaction, including turn-taking, role-based participation, constrained AI feedback, and persistent capture of intermediate work.

This represents a shift from *AI-supported collaboration* to *AI-constrained collaborative cognition*, where the system actively preserves the conditions required for reasoning.

3 Background and Learning Theory

This work is grounded in constructivist and social learning theory, which emphasize learning through active engagement and social interaction [6, 7].

This perspective aligns with social constructivist and distributed-cognition views of learning as an interaction among individuals, tools, and representa-

tions [7, 4], positioning AI as a participant in a broader cognitive system rather than a replacement for human reasoning.

The system's collaborative structure is informed by Process-Oriented Guided Inquiry Learning (POGIL), which organizes students around structured inquiry and defined collaborative responsibilities [2].

4 The Problem: LLMs and Cognitive Bypass

Traditional models of learning implicitly rely on a stable relationship between cognitive effort, reasoning processes, and observable output. In these models, students engage in cognitive effort, which produces reasoning, and this reasoning in turn generates observable artifacts such as written answers or code. Educators then evaluate these outputs as proxies for understanding.

The emergence of Large Language Models (LLMs) fundamentally disrupts this relationship. LLMs enable the direct generation of correct or plausible outputs without requiring the learner to engage in the underlying reasoning processes. As a result, the observable artifact—previously a proxy for cognition—can no longer be assumed to reflect the student's understanding.

This decoupling has significant implications for both learning and assessment. First, assessment signals degrade: correct answers no longer reliably indicate comprehension. Second, students may bypass productive struggle, a key mechanism through which cognitive restructuring occurs in constructivist learning theory. Third, opportunities for deep learning diminish, as the cognitive processes required for durable understanding are no longer consistently engaged.

We define this phenomenon as *cognitive bypass*: the ability to produce correct outputs without engaging in the reasoning processes those outputs are meant to reflect.

This challenge is not simply a matter of restricting AI usage, but of redesigning learning environments so that reasoning remains necessary for progress. Rather than treating AI as an external tool, we must embed constraints within the system itself that preserve the cognitive conditions required for learning. This shift is illustrated in Figure 1.

5 Design Principles

To address cognitive bypass, learning environments must enforce conditions in which reasoning is required for progress.

First, systems must require engagement in intermediate reasoning. Constructivist theory emphasizes the role of productive struggle and cognitive conflict in learning; therefore, systems should prevent direct answer generation

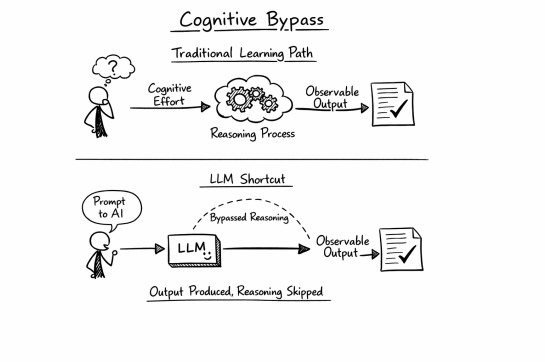


Figure 1: Cognitive bypass: LLMs weaken the connection between reasoning and observable output. In the traditional model (top), cognitive effort produces reasoning that leads to output. In the LLM-mediated case (bottom), output may be produced without engaging in the underlying reasoning process.

and instead require students to articulate partial explanations, predictions, or justifications before receiving feedback.

Second, collaboration must be structurally enforced. Social constructivist perspectives highlight that reasoning develops through interaction, yet collaboration is often uneven in practice. Mechanisms such as role assignment and turn-taking ensure that participation is distributed and visible.

Third, systems must preserve process. Because reasoning unfolds over time, capturing only final answers obscures how understanding develops. Systems should record intermediate responses, revisions, and feedback to support analysis of student thinking.

Fourth, AI must be constrained to a scaffolding role. While LLMs can provide useful guidance, unconstrained interaction allows them to replace rather than support reasoning. AI should instead prompt, question, and guide student reasoning without providing complete solutions.

Finally, systems must enforce participation. In group settings, some students may disengage or defer to others; requiring active contribution ensures that all participants engage in the reasoning process.

Together, these principles define a shift from permissive AI integration toward environments that actively preserve the cognitive conditions required for learning.

6 Persistent Epistemic Trace

To support reasoning as a first-class outcome, systems must capture not only final answers, but how those results are developed. We define a *persistent epistemic trace* as an append-only record of student responses, revisions, AI feedback, and executable artifacts.

Each submission produces a new entry rather than replacing prior work, preserving the sequence of reasoning over time. This enables instructors to examine how understanding develops and how participation is distributed within groups.

The trace provides instructors with process evidence for examining how answers developed and identifying possible cognitive bypass, while also supporting analysis of reasoning trajectories across attempts.

7 System Design: coLearn-AI

coLearn-AI is a web-based platform that operationalizes the design principles described above through a set of enforced interaction constraints. Rather than providing an open-ended interface to AI, the system structures how students interact with both each other and the AI, ensuring that reasoning remains central to the learning process.

Each student is assigned a fixed role for the duration of an activity, defining that student's responsibilities within the group. Submission control, however, rotates among group members: at any given time, only one student may submit the group's response. This combination preserves stable collaborative responsibilities while requiring every student to participate directly.

AI interaction is deliberately limited to a scaffolding role. Instead of generating solutions, the system uses AI to provide prompts, feedback, and probing questions that guide student reasoning without replacing it. The system supports multiple forms of student expression, including natural language responses, executable program code (e.g., Python and C++), and program output. AI-mediated feedback operates across these modalities, allowing the system to evaluate and guide not only written explanations, but also computational artifacts and their behavior. This multi-modal interaction is incorporated into the epistemic trace, enabling the system to capture reasoning as it unfolds across textual, procedural, and executable representations.

All interactions are persistently recorded as part of the system's epistemic trace, capturing not only final answers but also intermediate responses, revisions, and AI interventions. This ensures that the process of reasoning remains visible and analyzable.

Together, these constraints transform collaboration from an optional activity into a structured process enforced by the system itself.

7.1 Example Interaction

Consider an activity in which students examine a Python loop, predict its output, and explain how the loop variables produce that result. The activity is displayed to every member of the group, but only the student currently holding submission control can enter and submit the group's response. This structure requires the students to discuss and agree on an explanation before it is submitted.

Suppose the group correctly predicts the output but does not adequately explain how the loop variable changes. Rather than supplying the missing explanation, coLearn-AI might respond, "How does the value of `i` change during each iteration, and how does that change affect the value printed?" The group then discusses the prompt and submits a revised answer. This cycle continues until the response satisfies the instructor-defined criteria or the system determines that instructor intervention may be needed.

The original response, AI feedback, and subsequent revisions are retained in the interaction record, allowing instructors to examine how the group's reasoning developed. After the question is completed, submission control rotates to another group member.

A publicly accessible demonstration of coLearn-AI is available at `https://colearn-ai.com/demo/aied2026`.

8 Operationalizing POGIL as a Protocol

Process-Oriented Guided Inquiry Learning (POGIL) [2] provides one example of a structured, collaborative learning approach that can be expressed in terms of system-enforced interaction constraints. Rather than relying on instructor facilitation alone, these constraints can be encoded directly within the platform.

In this view, roles define stable responsibilities, structured activities guide inquiry, submission control rotates among participants, and participation is enforced through the system. While POGIL provides the immediate pedagogical model, the same approach generalizes to other models that require coordinated reasoning and controlled interaction.

9 Preliminary Observations

The system has been deployed extensively in three sections of an introductory programming course, with additional intermittent use in Data Structures, Ap-

plied Algorithms, and Software Development. These observations are informal and not derived from a controlled study, but they provide initial insight into how structured, AI-mediated collaboration affects student behavior.

Prior to this system, POGIL-style activities were conducted using shared Google Docs. In that setting, collaboration was often uneven, with one student frequently completing most of the work. A common pattern was "divide and conquer," where students split tasks rather than engaging in shared reasoning. Additionally, instructors had limited visibility into individual contributions and the overall group process.

Instructor observations suggested more consistently distributed participation with coLearn-AI. Turn-taking required students to alternate responsibility, while the interaction history gave instructors greater visibility into how answers developed. Students generally reported a positive experience with the system.

At the same time, new challenges emerged. Some students progressed more slowly and could delay group advancement, while others attempted to move ahead without full group alignment. These behaviors highlight the need for improved real-time awareness of group pacing and more refined mechanisms for balancing participation and progress.

These observations suggest that system-level constraints can influence collaborative behavior, but they also expose a design tension. AI guidance that is too restrictive may block progress, while overly permissive guidance may allow cognitive bypass. Similarly, fixed roles and rotating submission control distribute responsibility but can allow a slower participant to delay the group. Balancing reasoning, participation, and progress remains an open problem.

10 Future Work

Several directions for future work emerge from this system and its deployment.

First, controlled empirical studies are needed to evaluate the impact of structured, AI-mediated collaboration on learning outcomes. In particular, the epistemic trace provides a rich source of data for analyzing reasoning processes over time and comparing them across different instructional conditions.

Second, improving the design and management of AI guidance is a critical area for further development. This includes externalizing prompt definitions from system code, enabling reuse of course- and topic-level guidance, and developing tools that allow instructors to iteratively test and refine prompts more efficiently.

Finally, adaptive mechanisms for group formation, pacing, and participation balancing may help address observed challenges in coordinating collaborative work across students with different working speeds and engagement patterns.

11 Conclusion

Large Language Models fundamentally alter the relationship between cognitive effort, reasoning, and observable output, weakening traditional assumptions about how learning can be assessed and supported. This shift introduces the risk of cognitive bypass, where correct answers can be produced without engaging in the reasoning processes that underpin understanding.

In response, this work argues that preserving learning in AI-mediated environments requires system-level intervention. Rather than treating AI as an external tool, we embed constraints directly into the learning environment to ensure that reasoning remains necessary for progress. The coLearn-AI platform demonstrates how structured interaction, constrained AI guidance, and persistent epistemic traces can work together to support collaborative reasoning.

The central contribution of this work is a design approach that integrates learning theory, system constraints, and AI into a unified approach for computing education. While POGIL serves as one illustrative example, the approach generalizes to other structured pedagogical models that depend on coordinated reasoning and controlled interaction.

As AI continues to reshape educational practice, the challenge is not simply how to use these systems, but how to design environments in which they strengthen, rather than replace, the cognitive processes essential for learning.

References

[1] Imogen Casebourne et al. "Using AI to Support Education for Collective Intelligence". In: *International Journal of Artificial Intelligence in Education* 35.3 (2025), pp. 1597–1629. DOI: 10.1007/s40593-024-00437-7.

[2] David M. Hanson. *Instructor's Guide to Process-Oriented Guided-Inquiry Learning*. Pacific Crest, 2013.

[3] Wayne Holmes, Maya Bialik, and Charles Fadel. *Artificial Intelligence in Education: Promises and Implications for Teaching and Learning*. Center for Curriculum Redesign, 2019.

[4] Edwin Hutchins. *Cognition in the Wild*. MIT Press, 1995.

[5] OpenAI. "GPT-4 Technical Report". In: *arXiv* abs/2303.08774 (2023). DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs.CL].

[6] Jean Piaget. *The Origins of Intelligence in Children*. International Universities Press, 1952.

[7] Lev S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, 1978.

Latent Instructional Structure in CCSC Scholarship: A BERTopic Analysis of Proceedings, 2019–2025*

Dominic Wilson
Department of Computer Science
University of Findlay
Findlay, OH 45840
dominic.wilson@findlay.edu

Abstract

This study analyzes 682 papers from CCSC proceedings (2019–2025) using a BERTopic pipeline built on MPNet sentence-transformer embeddings, UMAP, and HDBSCAN. We identify 23 granular topics that hierarchically resolve into four instructional clusters, each organized by a coherent instructional logic. Analysis reveals what we term the Latent Instructional Structure (LIS) finding: three instructional dimensions operating in parallel. These are instructional content (what is being taught), instructional relationships (the pedagogical relationship between instructor, student, and institution), and instructional mode (the formal or applied treatment of content). Topics co-cluster when they share a dominant dimension value, even when topics share minimal keyword overlap. Longitudinally, the AI-related cluster grew from 21% of annual output in 2019 to 34% in 2025, driven by a Generative AI sub-cluster that reached 27% of output by 2025. Foundational CS instruction held stable in absolute volume over the same period, suggesting expansion rather than displacement. These findings demonstrate that transformer-based topic modeling can recover multi-dimensional instructional structure without document-level annotation, providing a scalable method for mapping and tracking pedagogical emphases over time.

1 Introduction

The Consortium for Computing Sciences in Colleges (CCSC) hosts ten regional conferences annually across the U.S., working to promote quality computer science education at the collegiate level [3]. These conferences produce peer-reviewed papers on pedagogical innovations, curricular developments, and trends in computing programs. Identifying overarching trends within this corpus offers useful insights for curriculum committees, accreditation bodies, and educational researchers. Previous bibliometric studies have successfully explored computing education trends, such as Green and Works' application of Non-Negative Matrix Factorization (NMF) to analyze topic distribution across CCSC regions [5], while other researchers have used Latent Dirichlet Allocation (LDA) to analyze curricular artifacts and instructional materials. For example, in wider educational contexts, Fiesler, Garrett, and Beard [4] analyzed a corpus of computing related ethics syllabi to characterize how ethical content is integrated across the curriculum, and Asmussen and Møller proposed a framework for using LDA topic modeling for literature review [1].

Despite this progress, prior work has not applied modern, transformer based topic modeling methods to the CCSC corpus, nor has it examined how pedagogical themes within this venue have shifted during a period of external disruptions. Between 2019 and 2025, undergraduate computing programs navigated both the pandemic and the rapid emergence of large language models (LLMs), which prompted widespread reconsideration of programming assessment and academic integrity. Mapping how CCSC publications reflect these shifts requires analytical methods capable of understanding semantic nuance beyond keyword frequency.

To accurately map how the computing curriculum adapted to these shifts at a national level, this study applies a state-of-the-art Natural Language Processing (NLP) pipeline to the CCSC corpus. We use a BERTopic architecture powered by deep sentence-transformer embeddings [6]. By clustering papers based on their underlying semantic similarities, we can extract pedagogical domains in an unsupervised manner [7, 12]. Unlike bag-of-words methods such as LDA or NMF, transformer-based embeddings encode semantic context and syntactic relationships, enabling the model to cluster documents that share instructional intent even when they share little surface vocabulary.

The objectives of this study are: (1) to establish a taxonomy of undergraduate computing curriculum as represented in the literature, (2) to conduct a longitudinal analysis tracking the relative prominence of these topics from 2019 to 2025, and (3) to evaluate whether the clustering structure recovers latent instructional dimensions that are not reducible to keyword overlap.

The remainder of this paper is structured as follows: The next section describes our data collection, preprocessing, and semantic clustering pipeline.

The Findings section presents the extracted topic taxonomy and their longitudinal trends. We interpret these trends and discuss what they imply for the discipline in the Discussion section and conclude with our summary of findings in the Conclusion.

2 Methodology

This study uses a multi stage topic modeling pipeline based on the BERTopic framework to study the latent structure of undergraduate computing education.

The analytical corpus comprises 682 full papers published in CCSC regional conference proceedings from 2019 to 2025 [2]. This range was selected because it corresponds to the span of CCSC proceedings for which complete digital PDF archives were publicly accessible at the time of analysis. Textual content was extracted from digital PDF archives, with reference sections removed to eliminate thematic noise. Preprocessing involved standard lemmatization and the application of a custom stopword list to filter out academic boilerplate and domain structural terms, ensuring that the resulting clusters reflected specific pedagogical content.

2.1 Semantic Clustering and Representation

The pipeline converts preprocessed documents into semantic embeddings using the all mpnet base v2 sentence transformer [11] selected from four candidate models based on topic coherence and dendrogram structure quality. Embeddings were reduced to five dimensions via UMAP [9] (n_neighbors =10, min_dist=0.0, cosine metric, random_state=42); the higher-dimensional projection preserves more embedding structure for density-based clustering than a standard two-dimensional reduction. HDBSCAN [8] (min_cluster_size =10, min_samples=5) then assigned each document to its dominant semantic neighborhood. Topic keywords were extracted using a CountVectorizer configured for unigrams and bigrams (ngram_range=(1,2), max_df=0.90) with a token pattern preserving hyphenated and underscore-joined domain terms. Following initial topic assignment, outlier documents were reassigned to their nearest topic using an embedding similarity strategy with a threshold of 0.3. The code and data supporting this analysis are available at `https://doi.org/10.5281/zenodo.20971911`.

2.2 Hierarchical Synthesis

To move from granular clusters to a high level taxonomy, Agglomerative Hierarchical Clustering [10] was applied to the topic level embeddings. This provided a structural scaffold, visualized in the dendrogram in Figure 1, which

was then refined through researcher-guided synthesis. Merges with clear pedagogical alignment were accepted; overrides were applied where keyword overlap masked meaningfully different pedagogical contexts. This process consolidated the 23 topics into four instructional clusters.

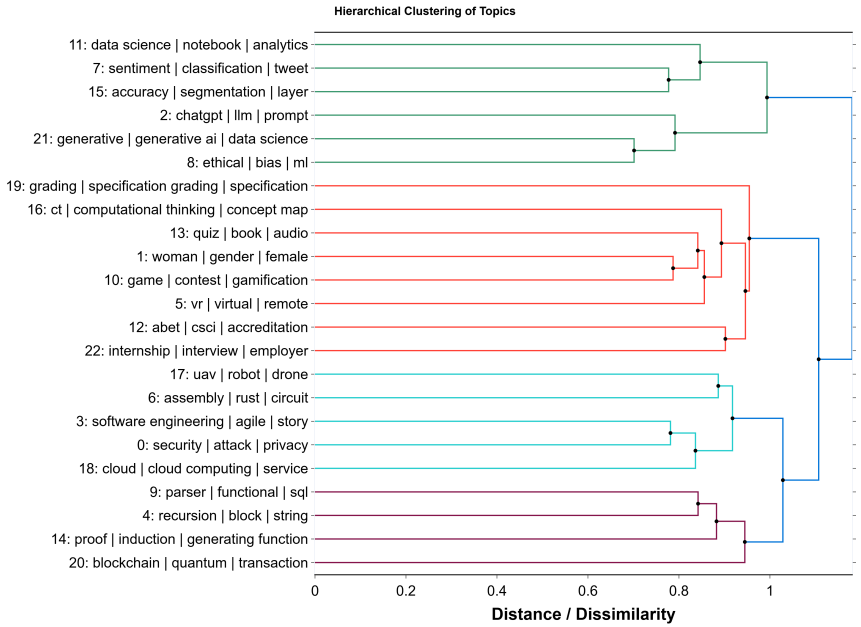


Figure 1: Dendrogram of Topics

3 Findings

The BERTopic pipeline applied to the 682-paper corpus yields 23 semantically coherent topics that hierarchically resolve into four clusters through agglomerative clustering of topic-level embeddings, designated Clusters A through D. The complete taxonomy, including sub-cluster assignments, topic labels, and representative keywords, is presented in Table 1. Longitudinal counts by cluster and sub-cluster are presented in Table 2.

Publication volume peaked at 118 papers in 2020, declined to a trough of 70 across 2021–2022 consistent with pandemic-related disruption to conference participation, and recovered thereafter, reaching 131 papers in 2025, an 11% increase over the previous peak.

Table 1: Topics and Their Clusters and Top Keywords

Cluster	Sub-cluster	Topic	Sample Keywords
A: AI, ML & Data Science Instruction	Applied ML, NLP & Deep Learning	11: Data Science & Notebook Instruction	data science, notebook, analytics, ml
		15: Deep Learning & Image Classification	accuracy, segmentation, deep learning
		7: Sentiment Analysis & NLP Classification	sentiment, classification, tweet, token
	Generative AI: Tools, Ethics & Literacy	8: AI Ethics, Bias & Fairness	ethical, bias, ml, fairness, generative
		21: Generative AI Literacy & Curriculum Design	generative ai, ml, fgcs, intelligence
		2: LLM Tools & Academic Integrity	chatgpt, llm, prompt, ai tool, plagiarism
B: Building & Operating Real Systems	Security, Cloud & Professional Software Engineering	3: Agile & DevOps Project Pedagogy	software engineering, agile, story, client
		18: Cloud Computing Instruction	cloud, cloud computing, service, storage
		0: Cybersecurity & IoT Security Instruction	security, attack, privacy, iot, threat
	Systems Architecture & Physical Computing	6: Systems Architecture & Simulation	assembly, rust, circuit, simulator
		17: UAV Navigation & Autonomous Systems	uav, robot, drone, marker, robotics
C: Foundational CS & Math. Reasoning	Crypto. & Adv. Theor. Sys	20: Cryptography & Blockchain Theory	blockchain, quantum, transaction, rsa
	Programming Languages, Logic & Proof	4: Core Programming Concepts & Tracing	recursion, block, string, animation, tracing
		9: Functional Programming & Language Theory	parser, functional programming
		14: Proof Writing & Discrete Mathematics	proof, induction, generating function
D: Pedagogy, People & Program Design	Engagement, Belonging & Outreach	16: Computational Thinking & K-12 Outreach	ct, computational thinking, concept map
		10: Game-Based & Novice Engagement	game, contest, gamification, workshop
		1: Gender, Belonging & Retention	woman, gender, female, accessibility
		5: Immersive & Mixed Reality Learning	vr, virtual, remote, ar, live-coding, spatial
		13: Introductory CS Engagement & Resources	quiz, book, c_plus_plus, collaborative
	External Accountab.: Accred. & Workforce	12: ABET Accreditation & Program Standards	abet, csci, accreditation, rationale
		22: Internships, Career Readiness & Workforce Prep	internship, interview, employer, readiness
	Grading Reform & Mastery Assessment	19: Grading Reform & Mastery Assessment	grading, specification grading, ungrading

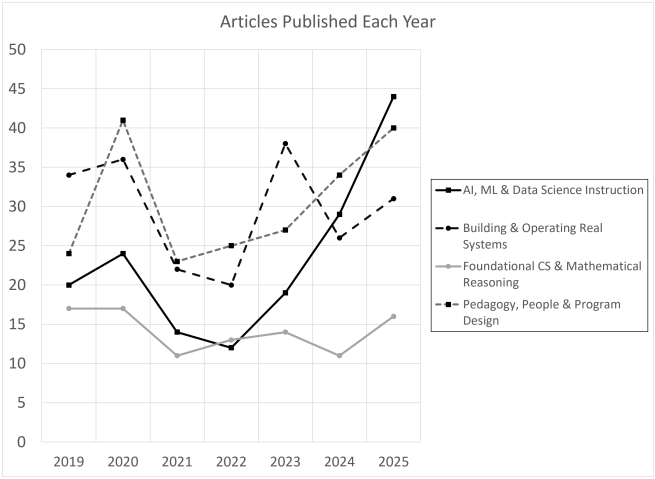


Figure 2: Annual Publications of Topic Clusters

3.1 The Four-Cluster Taxonomy

Cluster A, AI, ML & Data Science Instruction, consists of two sub-clusters: Applied ML, NLP & Deep Learning (Topics 7, 15, 11), in which students apply classification models, deep learning pipelines, and analytics workflows to real domain problems; and Generative AI: Tools, Ethics & Literacy (Topics 21, 8, 2), which encompasses three adjacent framings of generative AI (i.e. as curriculum content, as subject of ethical critique, and as active participant in the classroom). Cluster A is the fastest-growing cluster in the corpus, growing from 20 papers in 2019 to 44 in 2025.

Cluster B, Building & Operating Real Systems, groups five technically unrelated domains (systems architecture simulation, UAV navigation, cybersecurity, Agile software engineering, and cloud infrastructure,) across two subclusters: Security, Cloud & Professional Software Engineering (Topics 0, 3, 18) and Systems Architecture & Physical Computing (Topics 6, 17). The grouping provides evidence for the study's central claim: all five domains share the instructional mode of professional practice under real-world constraints.

Cluster C, Foundational CS & Mathematical Reasoning, consists of Programming Languages, Logic & Proof (Topics 4, 9, 14), which unites formal language theory, introductory programming mechanics, and discrete mathematics under a shared intent of making formal reasoning accessible, and Cryptography & Advanced Theoretical Systems (Topic 20), which occupies a peripheral position. Topic 20's assignment to Cluster C rather than Cluster B reflects

Table 2: Annual Distribution of Papers Across Clusters

	'19	'20	'21	'22	'23	'24	'25	Total
A: AI, ML & Data Science Instruction	**20**	**24**	**14**	**12**	**19**	**29**	**44**	**162**
Applied ML, NLP & Deep Learning	16	14	9	6	9	6	9	69
Generative AI: Tools, Ethics & Literacy	4	10	5	6	10	23	35	93
- LLM Tools & Integrity	*1*	*5*	*0*	*4*	*3*	*19*	*22*	*54*
- Ethics, Bias & Fairness	*2*	*3*	*3*	*2*	*6*	*3*	*7*	*26*
- Literacy & Curriculum	*1*	*2*	*2*	*0*	*1*	*1*	*6*	*13*
B: Building & Operating Real Systems	**34**	**36**	**22**	**20**	**38**	**26**	**31**	**207**
Security, Cloud & Prof. Software Eng.	24	27	18	16	31	18	28	162
Sys. Arch. & Physical Computing	10	9	4	4	7	8	3	45
C: Foundational CS & Math. Reasoning	**17**	**17**	**11**	**13**	**14**	**11**	**16**	**99**
Cryptography & Adv. Theoretical Systems	1	1	2	1	0	2	5	12
Prog. Languages, Logic & Proof	16	16	9	12	14	9	11	87
D: Pedagogy, People & Program Design	**24**	**41**	**23**	**25**	**27**	**34**	**40**	**214**
Engagement, Belonging & Outreach	21	31	18	17	24	25	28	164
External Accountability: Accred. & Workforce	3	7	2	7	2	5	10	36
Grading Reform & Mastery Assessment	0	3	3	1	1	4	2	14
Total	**95**	**118**	**70**	**70**	**98**	**100**	**131**	**682**

a mode-sensitive distinction: its papers engage cryptography as a mathematical object through formal proof and modular arithmetic, whereas Cluster B's cybersecurity papers engage security as applied threat-response practice.

Cluster D, Pedagogy, People & Program Design, is the largest cluster at 214 papers and consists of three sub-clusters: Engagement, Belonging & Outreach (Topics 1, 10, 13, 5, 16); External Accountability: Accreditation & Workforce Readiness (Topics 12, 22); and Grading Reform & Mastery Assessment (Topic 19). No topic in Cluster D is organized around a technical subject-matter domain (i.e., every keyword list is drawn exclusively from the vocabulary of pedagogical strategy, learner population, program evaluation, or institutional accountability), making it the most diagnostically significant cluster for the study's central claim.

3.2 The Latent Instructional Structure Finding

The four-cluster structure supports what we call the Latent Instructional Structure (LIS) finding: that transformer-based topic embeddings recover a multi-dimensional instructional structure in which distinct clustering dimensions operate in parallel. Three such dimensions are identifiable in this model. Instructional content, meaning what is being taught, organizes Cluster A, where all six topics position AI/ML as the primary subject. Instructional relationships, meaning the pedagogical relationships between instructors, students, internal and external institutions, organizes Cluster D, whose eight topics are defined exclusively by this dimension. Instructional mode, meaning the formal or ap-

plied treatment of content, explains the assignment of Topic 20 to Cluster C rather than Cluster B, separating mathematical cryptography instruction from applied security practice, and is illustrated by the three-way relationship among Topics 20, 0, and 3: Topics 20 and 0 share an instructional content (both teach systems with security implications) yet assign to different clusters because their modes diverge, while Topics 0 and 3 share almost no vocabulary yet co-cluster because both require students to build practical professional systems. The implication is that shared vocabulary does not predict co-clustering while the dominant instructional dimension does.

3.3 Longitudinal Trends

The longitudinal data in Table 2 reveal a curriculum in structural transition. The most consequential directional trend is the growth of AI-related content: Cluster A grew from 20 papers (21% of annual output) in 2019 to 44 papers (34%) in 2025, driven by the Generative AI: Tools, Ethics & Literacy sub-cluster, which grew from 4 papers (4%) to 35 papers (27%) over the same period; the steepest growth trajectory of any sub-cluster in the model, accelerating sharply after 2023 consistent with the public availability of ChatGPT in late 2022.

The annual distribution of papers across all four clusters is visualized in Figure 2, which illustrates the diverging trajectories of Cluster A and Cluster D against the relative stability of Clusters B and C over the study period.

Against Cluster A's growth, Cluster C held stable in absolute terms, ranging between 11 and 17 papers per year with no directional trend. Its proportional share contracted from 18% in 2019 to 12% in 2025 as AI-related content expanded, suggesting expansion rather than displacement: new instructional territory is being added to the CCSC scholarly agenda without a corresponding reduction in the foundational core.

4 Discussion

The findings reveal a CCSC scholarly community that is simultaneously stable and in rapid transition. The stability is structural: Clusters B and D, which together account for 62% of the corpus, have held relatively consistent volume across the study period. The transition is topical: the Generative AI sub-cluster has grown faster than any other instructional area in the corpus, reshaping the proportional distribution of scholarly attention without displacing existing areas of focus.

The most consequential finding for curriculum committees is the expansion pattern in Cluster A. The Generative AI: Tools, Ethics & Literacy sub-cluster grew from 4 papers in 2019 to 35 in 2025, representing a 775% increase. This

growth is concentrated in a single topic. Topic 2 (LLM Tools & Academic Integrity) rose from 3 papers in 2023 to 19 in 2024 and 22 in 2025, accounting for 54 of the sub-cluster's 93 papers and nearly the entire 2024 jump. By contrast, Topic 8 (AI Ethics, Bias & Fairness) remained comparatively stable across the period (2–7 papers per year, no directional trend), and Topic 21 (Generative AI Literacy & Curriculum Design) appeared at low volume before 2022 (1–2 papers per year through 2021) but grew only in 2025. The presence of AI-literacy scholarship prior to the public release of ChatGPT indicates that AI as a curricular concern preceded that disruption, even though the post-2022 volume surge is driven overwhelmingly by academic-integrity and LLM-tool work.

The three-way structure of the Generative AI sub-cluster, with AI positioned simultaneously as tool, subject of critique, and curriculum content, reflects the different instructional problems that faculty are navigating. Though academic integrity currently dominates by volume, there is a need for curriculum frameworks that address all three rather than treating AI integration as a single unified challenge.

The stability of Cluster C (Foundational CS & Mathematical Reasoning) within the 11–17 papers/year range, and no directional trend is a finding that warrants direct communication to curriculum planners. A concern commonly expressed in computing education discourse is that the pressure to incorporate applied and AI-related content will erode instruction in foundational areas such as discrete mathematics, programming language theory, and formal proof. The longitudinal data do not support this concern in the CCSC community: foundational instruction is holding its absolute volume while applied and AI-related instruction grows around it. The appropriate policy implication is not that foundational instruction is safe from pressure, but that the CCSC community has so far responded to new instructional demands through addition rather than substitution.

The LIS finding carries a methodological implication that extends beyond the CCSC corpus. The recovery of pedagogically coherent clusters from an unsupervised model, without any document-level annotation or predefined intent taxonomy, suggests that transformer-based embeddings encode instructional context as a stable feature of academic writing in CS education. If this finding generalizes to other CS education venues, BERTopic and related methods could serve as scalable tools for mapping the instructional landscape of a field, tracking how pedagogical emphases shift over time.

The External Accountability sub-cluster (ABET accreditation and internship/career readiness) deserves discussion. The co-clustering of these two topics, despite sharing no significant keyword overlaps, suggests that the scholarly community frames both as instances of the same institutional problem: demon-

strating to external audiences that the program produces capable graduates. This framing has practical implications for program design: accreditation and career readiness are often treated as separate administrative concerns, but the embedding structure suggests they are perceived by many faculty as two faces of the same accountability relationship.

5 Limitations

Several limitations qualify these findings. The corpus is restricted to a single publication venue, and the taxonomic and longitudinal patterns may not generalize to research-intensive venues. The LIS finding rests on the interpretability of topic clusters rather than document-level validation: the claim that clusters reflect instructional dimensions rather than shared publication conventions or methodological similarity cannot be confirmed without human raters assigning intent labels to individual papers and comparing those labels with cluster assignments.

Finally, the researcher-guided hierarchical synthesis introduces subjectivity; the two peripheral topic assignments, Grading Reform & Mastery Assessment and Cryptography & Advanced Theoretical Systems, carry the highest uncertainty in the taxonomy and should be interpreted with appropriate caution.

6 Conclusion

This study applied a BERTopic pipeline to 682 papers published in CCSC proceedings from 2019 to 2025, yielding a four-cluster taxonomy of undergraduate computing education scholarship. The taxonomy organizes 23 semantically coherent topics into four instructionally meaningful clusters: AI, ML & Data Science Instruction; Building & Operating Real Systems; Foundational CS & Mathematical Reasoning; and Pedagogy, People & Program Design.

Three findings stand out. First, the clustering structure provides evidence for our Latent Instructional Structure hypothesis: that transformer-based topic embeddings recover a multi-dimensional instructional structure organized by instructional content, instructional relationships, and instructional mode rather than by keyword overlaps alone. The grouping of five technically unrelated domains (cybersecurity, Agile software engineering, cloud infrastructure, system architecture simulation, and UAV robotics) into a single cluster defined by a shared instructional mode of practical professional education is a demonstration of this property.

Second, the longitudinal data reveal a curriculum in expansion. The Generative AI: Tools, Ethics & Literacy sub-cluster grew from 4 papers (4% of

annual output) in 2019 to 35 papers (27%) in 2025, driven by accelerating post-2022 growth in LLM tools use and their related integrity concerns.

Third, foundational instruction has remained stable in absolute volume across the study period, even as AI-related content has grown substantially. Cluster C held stable at between 11 and 17 papers per year with no directional trend, while its proportional share contracted from 18% to 12% as the overall corpus expanded. The CCSC community appears to be absorbing new instructional demands through addition rather than substitution, a pattern that curriculum committees should monitor as AI-related content continues to grow.

Future work should extend this analysis to additional venues and incorporate document-level annotation to validate the LIS interpretation.

References

[1] Claus Boye Asmussen and Charles Møller. "Smart Literature Review: A Practical Topic Modelling Approach to Exploratory Literature Review". In: *Journal of Big Data* 6.93 (2019), pp. 1–18. DOI: 10.1186/s40537-019-0255-7.

[2] Consortium for Computing Sciences in Colleges. *CCSC Past Digital Issues*. 2025. URL: https://www.ccsc.org/journal/past-issues/ (visited on 04/20/2026).

[3] Consortium for Computing Sciences in Colleges. *Consortium for Computing Sciences in Colleges*. 2025. URL: https://www.ccsc.org/ (visited on 04/15/2026).

[4] Casey Fiesler, Natalie Garrett, and Nathan Beard. "What Do We Teach When We Teach Tech Ethics? A Syllabi Analysis". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20. Portland, OR, USA: ACM, 2020, pp. 289–295. ISBN: 978-1-4503-6793-6. DOI: 10.1145/3328778.3366825.

[5] Nathan Green and Karen E. Works. "Are Research Trends in the Consortium for Computing Sciences in Colleges Regionalized?" In: *The Journal of Computing Sciences in Colleges* 38.3 (2022), pp. 40–50.

[6] Maarten Grootendorst. "BERTopic: Neural Topic Modeling with a Class-Based TF-IDF Procedure". In: *arXiv preprint arXiv:2203.05794* (2022). DOI: 10.48550/arXiv.2203.05794. arXiv: 2203.05794 [cs.CL].

[7] Li Ma et al. “AI-Powered Topic Modeling: Comparing LDA and BERTopic in Analyzing Opioid-Related Cardiovascular Risks in Women”. In: *Experimental Biology and Medicine* 250.10389 (2025). DOI: 10.3389/ebm.2025.10389.

[8] Leland McInnes, John Healy, and Steve Astels. “hdbscan: Hierarchical Density Based Clustering”. In: *Journal of Open Source Software* 2.11 (2017), p. 205. DOI: 10.21105/joss.00205.

[9] Leland McInnes et al. “UMAP: Uniform Manifold Approximation and Projection”. In: *Journal of Open Source Software* 3.29 (2018), p. 861. DOI: 10.21105/joss.00861.

[10] Fionn Murtagh and Pierre Legendre. “Ward’s Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward’s Criterion?” In: *Journal of Classification* 31.3 (2014), pp. 274–295. DOI: 10.1007/s00357-014-9161-z.

[11] Kaitao Song et al. “MPNet: Masked and Permuted Pre-training for Language Understanding”. In: *arXiv preprint* (2020). DOI: 10.48550/arXiv.2004.09297. arXiv: 2004.09297 [cs.CL].

[12] Sezer Uğuz and Cagatay Tulu. “Topic Modeling Analysis in the Field of Large Language Models with BERTopic (2020–2024)”. In: *2024 Innovations in Intelligent Systems and Applications Conference (ASYU)*. Ankara, Turkey: IEEE, 2024. DOI: 10.1109/ASYU62119.2024.10757169.

Multilingual Hate Speech Detection Using a Machine Learning and Deep Learning-Based Framework*

Busrat Jahan, Roshit Niraula,
Mansi Bhavsar, and Rushit Dave
Department of Computer Information Science
Minnesota State University, Mankato
Mankato, MN 56001
{busrat.jahan, roshit.niraula, mansi.bhavsar, rushit.dave}@mnsu.edu

Abstract

Hate speech detection in multilingual and noisy online environments remains challenging due to contextual ambiguity, low-resource languages, and class imbalance. This research work presents a two-stage classification framework using a traditional machine learning model (SVM) and a deep learning model (XLM-RoBERTa) for automated binary hate speech detection and multi-label subtype classification (Insult, Violence, Sexual, Racism, and Religious) with threshold optimization. Approximately 46,000 Bengali and Nepali social media comments were collected from Facebook and YouTube. XLM-RoBERTa achieved 92% accuracy for binary classification (Hate/NoHate) and macro/micro F1-scores of 0.91/0.90 in subtype multi-label detection. Comparatively, SVM achieved 91% accuracy with lower micro/ macro F1-scores of 0.86/0.85. Both models performed well on racism, while violence remained challenging for SVM, and sexual remained marginally lower subtype for XLM-RoBERTa. Additionally, XLM-RoBERTa showed fewer misclassifications in the confusion matrix curve and higher ROC-AUC value (0.97). Finally, the results indicate that the transfer-based XLM-RoBERTa is

more capable than SVM for nuanced hate speech detection, especially when contextual knowledge is needed.

Keywords: Hate Speech Detection, SVM, XLM-RoBERTa, Binary Classification, Multilingual NLP

1 Introduction

Nowadays, almost everyone uses social media platforms. Due to the vast use of social media, more offensive words are being used. 70% of Bangladeshi women between the ages of 15-25 reported complaining of cyberbullying in 2019 [1, 13]. In the digital era, the growth of multilingual, particularly code-mixed languages such as Hinglish, a blend of Hindi and English, has made it more difficult to detect hate speech. Approximately 60% of the global population uses the internet to communicate in multiple languages [13].

Devanagari script is widely employed in languages like Hindi, Marathi, and Nepali, making it more difficult to identify hate speech due to linguistic diversity, regional differences, and code-mixing practices [9]. Additional challenges include emojis, incomplete sentences, improper punctuation, and code-mixed Bangla-English text (Banglish) [5, 6]. Prior work [7] has used deep learning models: CNN, BERT, XLM-RoBERTa, MURIL, and IndicBERT, LSTM, Random Forest, and the traditional machine learning models: SVM, Naïve Bayes, Logistic Regression, and Decision Trees to detect hate speech. There is a lot of research on the English language but hate speech detection in the combination of Bengali and Nepali languages remain underexplored. Drawing on this research work, we set our research objectives. The objective of this research work is as follows:

- To build a unified multilingual dataset with the combination of Bengali ($\sim$35K) and Nepali ($\sim$11K) social media comments with a standardized multi-label schema.
- To develop a two-stage classifier detection: stage 1: binary (Hate/No-Hate) classifier and stage 2: multi-label subtype classification (Insult, Violence, Sexual, Racism, Religious).
- To evaluate both models performance for Bengali and Nepali Language hate speech detection using precision, recall, F1-score, accuracy, confusion matrix, and ROC-AUC.
- To address class imbalance and analyze the impact of contextual embeddings on models' performance across both architectures.

2 Literature Review

Hate speech detection has been a major research area in Natural Language Processing (NLP) and Machine Learning. Recent research shows that transformer-based and multilingual deep learning models have significantly replaced traditional machine learning models. Davidson et al. (2017) [3] collected 24,783 tweets and used lexical methods to classify hate speech, offensive language, or neutral content, highlighting that limited context might mix hate speech with generally offensive language [3]. Romim et al. (2021) [12] used the HS-BAN dataset across seven categories (sports, politics, etc.) and achieved an F1-score of 86.78% with a Bi-LSTM model [12]. Belal et al. (2023) [2] used 16,073 instances, categorized into six types: vulgar, hate, religious, threat, troll, and insult, to address Bengali toxic language detection and demonstrated that multi-label systems with BERT, CNN-BiLSTM, and attention mechanisms obtained 89.42% accuracy for binary and 78.92% accuracy & 0.86 F1-score for multi-label tasks, noting that binary tasks are simpler than multi-label classification [2]. Bade et al. (2024) [15] used the DravidianLangTech@EACL 2024 shared task dataset for code-mixed Telugu hate/offensive speech detection and found that TF-IDF with Random Forest achieved a macro-F1 of 0.492 [15], while Guragain et al. (2025) [7] studied Devanagari-script hate speech in Hindi and Nepali dataset and applied an ensemble of multilingual BERT models (XLM-RoBERTa, MURIL, IndicBERT) with backtranslation, obtaining F1-score: 0.6914 with facing class imbalance challenges [7]. Naseeb et al. (2025) [11] used the combination of machine learning (LR, SVM, RF, NB, KNN, and GBM) and deep learning (CNN, RNN, LSTM, and GRU) models and extended challenges in Roman Urdu due to a lack of standardized spelling [11]. All these recent works indicate strong progress in multilingual and low-resource hate speech detection, but no prior work has found on a combined Bengali and Nepali dataset.

3 Methodology

The proposed methodology for this work is as follows in Figure 1.

3.1 Dataset

Our experiments are conducted on a multilingual hate speech dataset combining Bengali and Nepali social media comments. The Bengali dataset (∼35,000 entries) was collected from Facebook and YouTube through the publicly available 'Hugging Face' platform [14], while the Nepali dataset (∼11,000 entries) was collected from high-subscriber YouTube channels within the News and Politics domain [10]. Both datasets follow the same multi-label schema, enabling

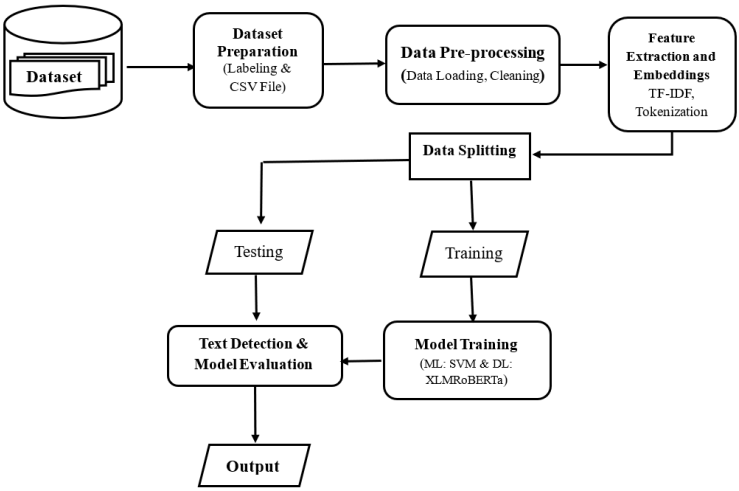


Figure 1: Proposed Workflow

seamless integration into a unified experimental resource.

3.2 Data Preparation and Preprocessing

Initially, the Nepali dataset was not labelled. However, we used lexicon cleansing, category mapping, hate-speech tagging, dataset validation, and multilingual merging for labelling Nepali dataset. Lexical mappings are aligned entries into five standardized categories: insult, violence, sexual, racism, and religious, alongside a binary Hate/NoHate label. We implemented the following preprocessing steps: Text was normalized using Unicode normalization(NFKC); To maintain contextual structure, replaced URLs with [URL] and User mentions (@user) with [USER]; To normalize whitespace and token boundaries, removing the hashtag (#) symbol while preserving the keyword for semantic relevance; To remove sentence separators, such as repeated ‘danda‘ symbols, emojis artifact, duplicate entries, invisible/control characters, such as zero-width spaces and byte-order markers; To normalize repeated characters, punctuation (such as, ‘!!¡ → ‘¡) and filtered out extremely short or empty with word truncation.

3.3 Feature Extraction

For the machine learning model, we used word-level TF-IDF with unigrams and bigrams and character-level TF-IDF (3-5 grams), concatenating both semantic information and subword structures. We set the max_length for the word count at 300 because TF-IDF does not benefit from extremely long texts. For the deep learning model, tokenization on the cleaned text used the pretrained XLM-RoBERTa tokenizer, which performed subword tokenization using SentencePiece. Each input text was converted into input IDs and attention masks, with dynamic padding and truncation with a fixed maximum sequence length of 128 tokens.

3.4 Model Architecture

For Machine Learning Model (SVM): We used text classification based on TF-IDF features with a linear SVM in a One-vs-Rest configuration for a multi-label dataset. Two complementary feature extractors were used to transform text data into numeric values. Word-level TF-IDF with unigrams and bigrams (max features=20,000, min document frequency=2) encoded semantic and syntactic patterns, while character-level TF-IDF with 3–5 grams using (char_wb) to handle noisy, multilingual, and morphologically complex text.

For Deep Learning Model (XLM-RoBERTa): Similarly, a transformer-based multilingual encoder XLM-RoBERTa was used to generate contextual representations. It was also formulated as a two-stage classification pipeline with a shared-backbone to optimize for both tasks (class imbalance & multi-label prediction). A layer-wise learning rate decay strategy was employed, with higher layers and task-specific heads receiving higher learning rates, while lower layers were used for embeddings to preserve general multilingual knowledge. Two-stage heads were used, a binary classifier(softmax) for hate vs nohate detection, and a multi-label classifier(sigmoid) to predict specific hate speech subtypes.

3.5 Data Splitting

For the machine learning model, the dataset was divided into 80% training (36,984) sets and 20% testing (9247). Alternatively, the deep learning model was trained on 80% of the dataset (36984), 10% for validation (4623) and 10% for testing (4624) using stratified sampling to preserve class distribution.

3.6 Training Process

For Machine Learning Model (SVM): The training process started with standard text preprocessing, including cleaning, followed by TF-IDF vectorization

at both the word and character levels. The combined feature matrix was used to train a One-vs-Rest Linear SVM model, which individually trained a binary classifier for each label in the multi-label setup. The model used class balancing (class_weight='balanced') to handle label imbalance in the dataset. Model validation was performed using 10-fold cross-validation on the training set with micro-average F1 score as the evaluation metric.

For Deep Learning Model (XLM-RoBERTa): The training process started with standard text preprocessing, including cleaning and tokenization, followed by a two-stage sequential approach. In stage 1, a binary classifier was trained using weighted cross-entropy loss with label smoothing (0.1) to handle class imbalance. In stage 2, a subtype classifier was trained only on hate-labelled samples using a multi-label setup with a focal loss (loss = $(1 - p_t)^{\gamma} * \text{BCE}$) objective, which emphasized harder and rarer examples by down-weighting easy predictions. Both stages used AdamW optimizer (weight decay = 0.01, learning rate =2e5), 5 epochs, cosine scheduling with warmup ratio (0.06), and GPU support. Finally, softmax was used for binary classification, while sigmoid with a threshold of 0.5(Values ≥ 0.5 are assigned label 1 and Values $<$ 0.5 are assigned label 0) was applied for multi-label subtype predictions. The final evaluation included precision, recall, F1-score (micro and macro), and Hamming loss.

3.7 Evaluation Metrics

For machine learning and deep learning models, we have evaluated performance using: Precision; Recall; F1-score; Micro-averaged F1; Macro-averaged F1. Additionally, we have used confusion matrices and ROC-AUC curves also. Additionally, for the machine learning model, we assess robustness using 10-fold cross-validation on the training split, with micro-F1 as the primary metric.

4 Results

4.1 For Machine Learning Model (SVM)

The SVM model has achieved a Macro Avg-F1 of 0.85, and the Micro-F1 score of 0.86 confirms strong overall multi-label performance. Table 1 shows notable variation in the class-wise metrics difficulty across categories. Racism achieves the highest precision (0.97) and F1-score (0.93), suggesting clear lexical cues and minimal false positives. In contrast, Violence exhibits lower precision (0.68) and F1-score (0.71), indicating overlap with other aggressive or emotional expressions. Sexual content and the general Hate/NoHate label demonstrate balanced precision, recall, and F1-score while minority categories benefit from the use of class-weight adjustments during training.

Table 1: Overall Metrics Performance

List	Precision	Recall	F1-score	Support
Hate/NoHate	0.89	0.86	0.87	3440
Insult	0.85	0.85	0.85	2555
Violence	0.68	0.74	0.71	654
Sexual	0.90	0.86	0.88	1326
Racism	0.97	0.90	0.93	430
Religious	0.86	0.84	0.85	592
micro avg	0.86	0.85	0.86	8997
macro avg	0.86	0.84	0.85	8997
weighted avg	0.87	0.85	0.86	8997
samples avg	0.33	0.31	0.32	8997

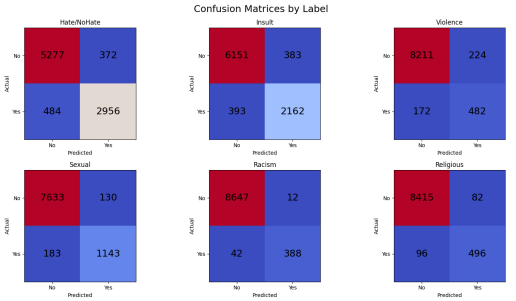


Figure 2: Confusion Matrix of all Categories

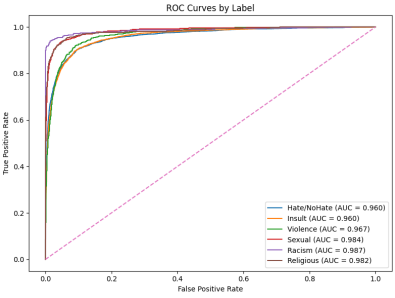


Figure 3: ROC-AUC curve

```
Your work is very good.
Raw: तपाईको काम एकदम राम्रो छ।
Cleaned: तपाईको काम एकदम राम्रो छ

Decision scores (higher = more confident):
  Hate/NoHate: -1.049
  Insult: -1.193
  Violence: -1.958
  Sexual: -1.189
  Racism: -1.294
  Religious: -1.712

Predictions:
  Hate/NoHate: 0
  Insult: 0
  Violence: 0
  Sexual: 0
  Racism: 0
  Religious: 0

Summary: NOT HATE
```

```
I will kill him.
Raw: ওকে মেরে ফেলবো।
Cleaned: ওকে মেরে ফেলবো

Decision scores (higher = more confident):
  Hate/NoHate: -0.191
  Insult: -0.774
  Violence: 0.380
  Sexual: -0.810
  Racism: -1.300
  Religious: -1.240

Predictions:
  Hate/NoHate: 1
  Insult: 0
  Violence: 1
  Sexual: 0
  Racism: 0
  Religious: 0

Summary: HATE
Categories: Violence
```

Figure 4: Example of Hate/NoHate Detection Using SVM

Figures 2 and 3 demonstrate that the model maintains high classification accuracy, with the ROC-AUC accuracy of 0.96 across all categories. Figure 4 shows that our SVM-based machine learning model detected the Hate/NoHate speech correctly.

4.2 For Deep Learning Model (XLM-RoBERTa)

In Table 2 shows final test set evaluation for binary stage with 0.92 accuracy.

Table 2: Final test set evaluation for binary stage

List	Precision	Recall	F1-score	Support
NoHate	0.93	0.94	0.93	2828
Hate	0.90	0.88	0.89	1727
accuracy			0.92	4555
macro avg	0.91	0.91	0.91	4555
weighted avg	0.91	0.91	0.91	4555

Table 3 shows the final Subtype stage evaluation results, with a micro/macro-avg F1-score of 0.90/0.91, indicating balanced precision and recall across classes, and a Hamming loss of 0.0632. Racism performs best (F1-score = 0.96), Sexual performs slightly lower (F1-score = 0.85). The weighted avg F1-score of 0.90 reflects good performance considering weight imbalance, though the lower samples average at (F1 = 0.85).

Table 3: Final test set evaluation for subtype stage

List	Precision	Recall	F1-score	Support
Insult	0.90	0.96	0.93	1291
Violence	0.88	0.85	0.87	332
Sexual	0.83	0.87	0.85	664
Racism	0.99	0.93	0.96	228
Religious	0.96	0.90	0.93	312
micro avg	0.89	0.92	0.90	2827
macro avg	0.91	0.90	0.91	2827
weighted avg	0.89	0.92	0.90	2827
samples avg	0.86	0.86	0.85	2827

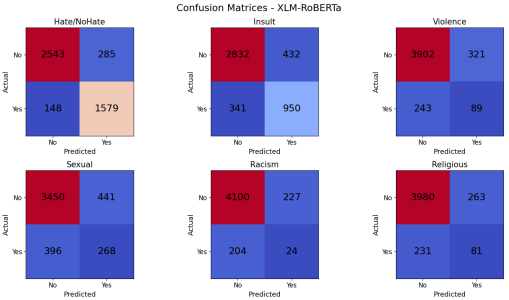


Figure 5: Confusion Matrix of all Categories

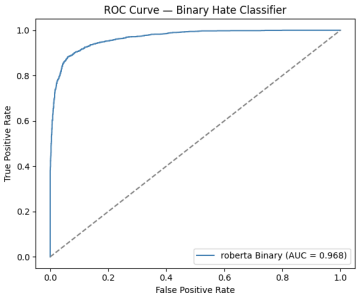


Figure 6: ROC-AUC curve

	text	english_translation	Hate_prob	Insult_prob	Insult	Violence_prob	Violence	Sexual_prob	Sexual	Racism_prob	Racism	Religious_prob	Religious	Hate/NoHate	predicted_type
0	तपाईंको काम एकदम राम्रो छ।	Your work is very good.	0.077200	0.465000	0	0.103900	0	0.297600	0	0.238100	0	0.163400	0	0	no_hate
1	चोर जेरी भुक्न थालिस	The thief started barking again.	0.955900	0.624100	1	0.162500	0	0.254400	0	0.143300	0	0.061500	0	1	Insult
2	তুই একটা বোকা।	You are an idiot.	0.911400	0.652400	1	0.076700	0	0.198400	0	0.130500	0	0.096400	0	1	Insult
3	তোমার ভিডিওটা খুব ভালো লেগেছে।	I really liked your video.	0.069800	0.442100	0	0.061500	0	0.433000	0	0.281800	0	0.072800	0	0	no_hate

Figure 7: Example of Hate/NoHate Detection Using XLM-RoBERTa

Figures 5 and 6 show that the model maintains high classification accuracy; the ROC-AUC accuracy is 0.968(∼0.97) across all categories. Figure 7 shows an example of Hate/NoHate classification, and it perfectly detects both stages.

4.3 Comparative Analysis

In our machine learning model (SVM) and deep learning model (XLM-RoBERTa), we achieved the following accuracies (in Table 4) across two languages (Bengali and Nepali) for each class:

Table 4: Comparative Analysis SVM vs XLM-RoBERTa

ML model (SVM)	**DL model (XLM-RoBERTa)**
Hate/NoHate: Accuracy=0.9064	Hate/NoHate: Accuracy=0.9164
Insult: Accuracy=0.9145	Insult: Accuracy=0.9082
Violence: Accuracy=0.9567	Violence: Accuracy=0.9704
Sexual: Accuracy=0.9657	Sexual: Accuracy=0.9477
Racism: Accuracy=0.9941	Racism: Accuracy=0.9945
Religious: Accuracy=0.9805	Religious: Accuracy=0.9853

5 Discussion

The results indicate that both models (SVM and XLM-RoBERTa) are effective for hate speech detection, although XLM-RoBERTa showed a slightly better overall performance. In the binary classification task, XLM-RoBERTa achieved strong and well-balanced performance, with precision, recall, and F1-score all ranging from 0.91 to 0.92, indicating reliable classification between hate vs no-hate. For subtypes, both models performed especially well on racism (SVM F1= 0.93), (XLM-RoBERTa F1=0.96). By contrast, violence (F1=0.71) was a more difficult category, particularly for SVM, while sexual (F1=0.85) generated comparatively lower scores for XLM-RoBERTa, indicating more complex and overlapping linguistic patterns. According to previous research, contextual embedding improved performance in ambiguous situations, with XLM-RoBERTa demonstrating a small improvement across various classes [4]. Additionally, SVM remains competitive, especially for classes with clear lexical indicators [8], but results are affected by class imbalance, with minority classes showing more variance. Overall, SVM is efficient and reliable, transformer-based models (XLM-RoBERTa) are more appropriate for detecting hate speech in complex situations.

6 Conclusion and Future Work

This quantitative study indicates that the deep learning model (XLM-RoBERTa) outperforms the traditional machine learning model (SVM) for the automated hate speech detection in Bengali and Nepali social media comments (Facebook and YouTube). In the binary stage, it achieves an F1-score = 0.91 with 92% accuracy and in the subtype stage, it achieves micro/macro F1-scores of 0.90/0.91 and a Hamming Loss of 0.0632, showing robust and balanced performance across categories. In comparison, SVM achieves an F1-score of 0.87 with lower subtype stage performance (micro F1=0.86, macro F1=0.85). Only violence (SVM) and sexual (XLMRoBERTa) subtype categories have lower scores for both models. Overall, the transformer-based model (XLM-RoBERTa) is more robust and accurate for hate speech detection. A major limitation of this work is the class imbalance in the datasets used for both binary & subtype classification stages, which is likely influenced the performance of both models. Future research should test the models on larger and more balanced datasets to improve generalization and explore ensemble methods, multilingual datasets, and additional transformer architectures for comparison.

References

[1] Md. Faisal Ahmed et al. "Cyberbullying Detection Using Deep Neural Network from Social Media Comments in Bangla Language". In: *Proceedings of the 2nd International Conference on Advances in Computing, Communication, Control and Networking (ICAC3N)*. 2021, pp. 1162–1167. DOI: 10.1109/ICAC3N53548.2021.9725514. URL: https://doi.org/10.1109/ICAC3N53548.2021.9725514.

[2] Tanveer Ahmed Belal, G. M. Shahariar, and Md. Hasanul Kabir. "Interpretable Multi Labeled Bengali Toxic Comments Classification using Deep Learning". In: *Proceedings of the 3rd International Conference on Electrical, Computer and Communication Engineering (ECCE 2023)*. 2023. DOI: 10.1109/ECCE57851.2023.10101588. URL: https://doi.org/10.1109/ECCE57851.2023.10101588.

[3] Thomas Davidson et al. "Automated Hate Speech Detection and the Problem of Offensive Language". In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 11. 1. 2017. DOI: 10.1609/icwsm.v11i1.14955. URL: https://doi.org/10.1609/icwsm.v11i1.14955.

[4] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Vol. 1. 2019, pp. 4171–4186. DOI: 10.48550/arXiv.1810.04805. URL: https://doi.org/10.48550/arXiv.1810.04805.

[5] Noyon Dey et al. "Using machine learning to detect events on the basis of bengali and banglish facebook posts". In: *Electronics* 10.19 (2021), p. 2367. DOI: 10.3390/electronics10192367. URL: https://doi.org/10.3390/electronics10192367.

[6] Sayani Ghosal et al. "Inculcating Context for Emoji Powered Bengali Hate Speech Detection using Extended Fuzzy SVM and Text Embedding Models". In: *ACM Transactions on Asian and Low-Resource Language Information Processing* (2023). DOI: 10.1145/3589001. URL: https://doi.org/10.1145/3589001.

[7] Anmol Guragain et al. "NLPineers@ NLU of Devanagari Script Languages 2025: Hate Speech Detection using Ensembling of BERT-based models". In: *Proceedings of the 1st Workshop on Challenges in Processing South Asian Languages (CHiPSAL 2025)*. 2025, pp. 320–326. URL: https://github.com/Anmol2059/NLPineers.

[8] Thorsten Joachims. "Text categorization with Support Vector Machines: Learning with many relevant features". In: *Proceedings of the European Conference on Machine Learning (ECML)*. 1998, pp. 137–142. DOI: 10.1007/BFB0026683. URL: https://doi.org/10.1007/BFB0026683.

[9] Kengatharaiyer Sarveswaran et al., eds. *Proceedings of the 1st Workshop on Challenges in Processing South Asian Languages (CHiPSAL 2025)*. Abu Dhabi, UAE: International Committee on Computational Linguistics, 2025. URL: https://sites.google.com/view/chipsal.

[10] Oyesh Mann Singh et al. "Aspect Based Abusive Sentiment Detection in Nepali Social Media Texts". In: *Proceedings of the 17th International Conference on Natural Language Processing (ICON)* (2020). URL: https://github.com/kldtz/bratiaa.

[11] Amna Naseeb et al. "Machine Learning- and Deep Learning-Based Multi-Model System for Hate Speech Detection on Facebook". In: *Algorithms* 18.6 (2025). DOI: 10.3390/a18060331. URL: https://doi.org/10.3390/a18060331.

[12] Nauros Romim et al. "HS-BAN: A Benchmark Dataset of Social Media Comments for Hate Speech Detection in Bangla". Preprint, accessed Apr. 20, 2026. 2021. URL: https://github.com/strohne/Facepager.

[13] Yash Shukla et al. "A Multilingual BERT-Based Framework for Robust Online Hate Speech Detection". In: *Proceedings of the 2024 IEEE International Conference on Computer Vision and Machine Intelligence (CVMI)*. 2024. DOI: 10.1109/CVMI61877.2024.10782783. URL: https://doi.org/10.1109/CVMI61877.2024.10782783.

[14] Sumaiya Afroze. *Multi-Label Bangla Hate Speech Data*. Hugging Face. 2023. URL: https://huggingface.co/datasets/sumaiya-afroze/Multi-Label_Bangla_Hate_Speech_Data.

[15] Girma Yohannis Bade et al. *Social Media Hate and Offensive Speech Detection Using Machine Learning Method*. 2024. URL: https://codalab.lisn.upsaclay.fr.

Assessment in Data Structures: Four Approaches*

James Vanderhyde and Jean Mehta
Computer Science
Saint Xavier University
Chicago, IL 60655
{vanderhyde,mehta}@sxu.edu

Abstract

Over the course of four years, we modified the structure and assessment of our Data Structures and Algorithms course. This paper documents our travels on this adventure. We variously used traditional grading, oral exams, competency-based grading, and a portfolio model. Each model has strengths, and we try to combine the strengths to obtain the best possible model. The final goal is a two-pronged focus on content and metacognition.

1 Context

The data structures course has long been the lynchpin of undergraduate computer science curricula[5]. We call our course CMPSC 311, Data Structures and Algorithms. It is the third course in a sequence of Java programming courses. The content includes classical data structures topics including linked lists, stacks, queues, hash tables, and binary search trees. The course is taught every fall semester to 25 to 50 students in one or two sections. We teach at a small liberal arts college in an urban environment.

The course procedures and assessments have changed over the years in a desire to incorporate authentic, equitable grading practices[3] and to compensate for the effects of online tools such as Chegg and generative AI.

2 The Four Approaches

2.1 Traditional Grading

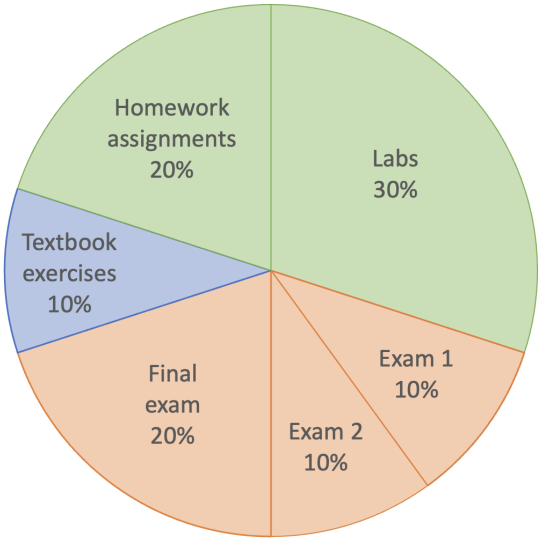


Figure 1: Traditional grading pie chart

Before the pandemic, 50% of the course grade came from labs and homework (usually programming problems). The rest was 20% from 2 tests, 20% from the final exam, and 10% electronic textbook exercises. In 2020 the course switched to online (because of the pandemic), and evaluation remained the same.

However, in 2021 and 2022, as the course remained online, there was rampant student use of online help of little pedagogical value (copying answers from Chegg, asking ChatGPT for answers) on the labs and homework. This was difficult to control because the course was online. The fact that 50% of the grade reflected how well students could copy and paste answers was greatly concerning.

2.2 Oral Exams

At about this time, one member of our advisory board suggested we were not preparing our students for technical interviews. Considering this along with the observed unhelpful habit of copying to get a grade, we switched to using oral exams as a means of testing competence in the material. We called

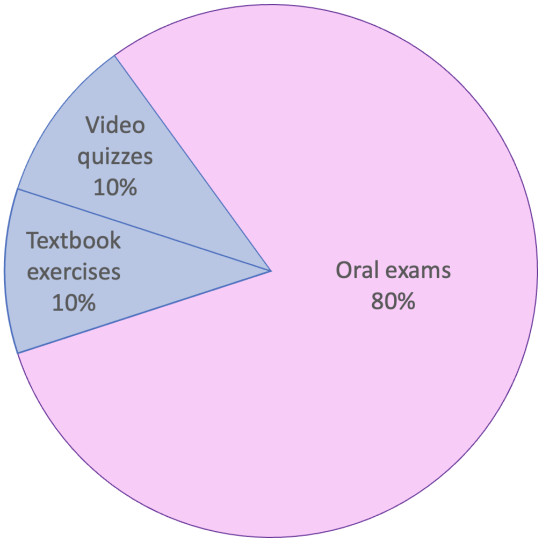


Figure 2: Oral exams pie chart

them technical interviews so that students would understand the connection to career readiness. 20% of the grade was from quizzes embedded in videos and exercises from the electronic textbook, the other 80% from technical interviews. The labs and homework assigned in previous years were still assigned but not graded. Instead, the solutions were published. It was up to the student to attempt the assignment and check the solution. Every two weeks, each student had a one-hour meeting with the instructor. This was during the pandemic lockdown, so it provided some essential instructor-student interaction. The meeting included a 30-minute interview to determine the student's level of competence with the material. Students had to achieve a C or better in each interview. Some students had to repeat the interview, others aced it. This process involved a lot of instructor time, but the course was online and there was no other grading. It was feasible for a small class, but it would not work for more than 20 students.

The oral exams quickly revealed that most of our students could not pass oral exams. It seemed they had learned very little about Java programming, even after a full year in Java coursework. Some students could not engage in oral exams at all, due to the stress. In the future, if we go with oral exams again, we will introduce them earlier in the curriculum. The two semesters of programming courses before Data Structures can have low-stakes oral exams

so the students learn to do it without breaking down. Perhaps with decreasing enrollments, the time commitment for instructors to give oral exams will become feasible again.

2.3 Competency-based Grading

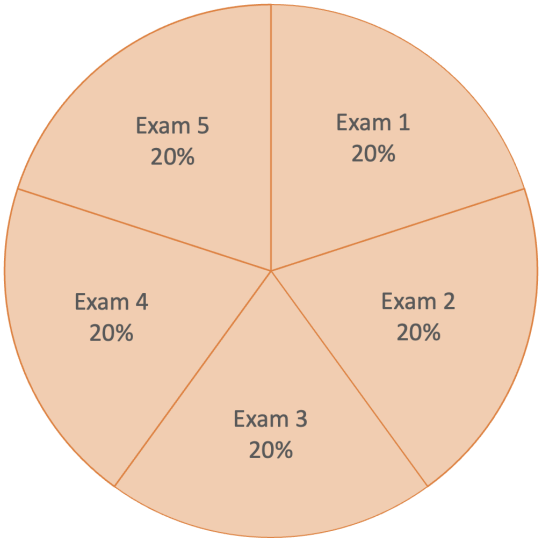


Figure 3: Competency-based grading pie chart

In Fall 2024, the course was back on campus and there were two sections with two different instructors. There was a lot of excitement in the computing education community around competency-based grading around this time[7][8][4], so we abandoned the technical interviews but maintained the competency model. The course grade was based 100% on five tests. Students must pass every test at 70% or above, and students may retake any test to improve their grade. The overall course grade was the average of the five tests.

Each test was a combination of a sequence of short-answer, concept-driven questions and a programming problem. The short-answer portion was closed book. The programming problem was done in a controlled online environment that recorded every action taken by the test taker. Starter code was provided, as well as relevant examples from coursework for reference. Both portions were done in a single session in a proctored classroom.

After each test, the instructor spent a lot of time checking if the code was

pasted in for the programming portion. In case it was, the student was given a 0 and required to retake the test. No one confessed to copying the answer from AI, but no one debated the accusation, either.

Homework assignments were not included in the course grade, but the instructor gave feedback on all homework, just as if it was graded. The purpose of this policy was to disincentivize copying answers on the homework and instead encourage students to work out the problems themselves and receive productive instructor feedback. However, the result was that most students turned in little to no homework. It seems they could not get past the concept of doing work that did not directly affect their grade. Some students said they did the homework but didn't bother to turn it in because it wasn't graded. Besides not doing homework, the biggest drawback of our approach this year was that some students found the high-stakes tests very stressful, even though retakes were allowed. It felt like a final exam every 3 weeks. The instructors believe that the test grades under such stressful conditions were not an accurate reflection of student learning.

2.4 Portfolio

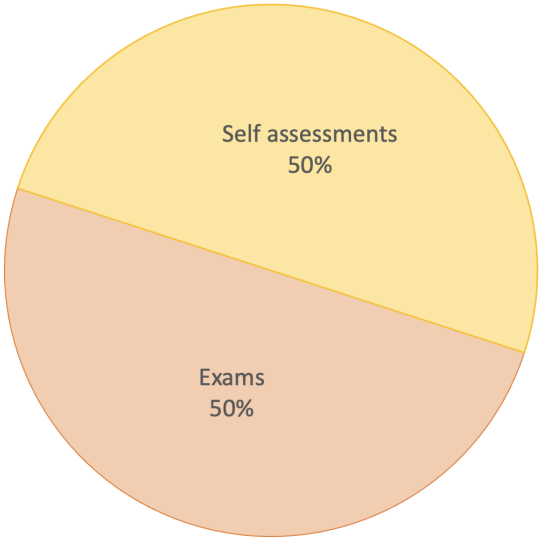


Figure 4: Portfolio pie chart

When the entire grade was based on tests, the students were stressed out.

In Fall 2025, to alleviate some of this stress, we decided to use a portfolio approach[1][6]. In this kind of approach, each student must demonstrate competence in the objectives of the course through summative assessments and a collection of formative assessments combined with student self-assessments. We balanced 50% of the grade from summative assessments (exams) with 50% from student self-assessments (we called them summaries).

Every one to two weeks, the students rated themselves on several outcome dimensions, and they used their grades on homework and quizzes as evidence. In one semester, we had seven exams and nine summaries. Homework was required and graded by the instructor, but it was not part of the course grade. Instead, students used their grades on e-textbook exercises, labs, and homework as evidence in their summaries.

Here is one example of a summary:

> These are the Student Learning Outcomes (SLOs) for Stacks and Queues. For each SLO, evaluate your current level of competency as Beginning, Developing, Proficient, or Expert. Use this as a tool for reflection and improvement.
>
> Stacks
>
> 1. Define the stack ADT.
> 2. Describe the push and pop operations.
> 3. Implement a stack class using an array.
> 4. Implement a stack class using a linked list.
> 5. Use the Stack class in Java.
>
> Queues
>
> 1. Define the queue ADT.
> 2. Describe the enqueue and dequeue operations.
> 3. Implement a queue class using an array.
> 4. Implement a queue class using a linked list.
> 5. Use the Queue interface in Java.
>
> My scores so far:

Assignment	My score	Possible score
Z7		10
Z7b		10
H6		40
L5		10
L5b		20
Z8		10
Z8b		5
Total		105

I would give myself an overall grade of _____ for this work.

Rationale: Explain why you assessed your learning (Beginning, Developing, Proficient, Expert) in the way that you did.

Upon reflection, what might you have done differently to improve your learning?

What are you planning to do (if anything) to prepare for the upcoming test on stacks and queues?

In the above example, note that students are first informed of the proposed learning outcomes for the section on stacks and queues. They are asked to rate themselves on each outcome. They are then asked to document their grades from exercises in our electronic textbook (Z7, Z7b, Z8, Z8b), classroom exercises (L5, L5b), which were group exercises using either pseudocode or diagrams, and a homework coding exercise (H6) which was practice using Java's built-in Stack class. These were all low stakes assignments. On the homework, students may have used AI despite being told not to. The instructor sometimes asked students to meet for code review to verify that students could defend and explain their code so that even if they copied it, at least they understood it.

This summary was submitted along with the student's self-grade (A, B, C, or F). The instructor carefully read the summary and self-grade and then discussed all this with the student. Most often, they both agreed on the grade. Occasionally a student graded more harshly than the instructor, but there was no instance of the student believing that they deserved a higher grade. The instructor was aware of possible bias in student self-evaluation (particularly by gender) and was watching for it, but we did not observe anything like that.

An unexpected benefit of listing the student work in a single summary was that it made it more obvious to the student that these are the points they earned for this topic. Without the summary, the grades are just miscellaneous items in a sea of grades in the LMS.

This semester the instructor was pleased with the outcome. There were

no student complaints of high anxiety or inability to take a test under these conditions. This in turn gave the instructor more confidence in the outcome of the tests and that we were measuring the students' knowledge and not their anxiety levels. Furthermore, the problem of students not completing homework was solved by requiring the students to report their homework scores on the summary. Finally, reliance on AI and other tools was reduced because the focus was turned toward learning rather than completion.

3 The Next Approach

This course is taught every fall, and we are gearing up for the next iteration of our grading scheme. We plan to keep the current split: 50% on tests, 50% on summaries. Tests give students the opportunity to demonstrate their knowledge at the conclusion of the topic, and summaries provide students with low stakes exercises to demonstrate their learning in the weeks prior to the test.

The instructors are still honing their skills at using the portfolio model. These are our plans:

- Keep the focus on the learning objectives for each topic. This develops the students' metacognition by requiring them to consider what they should have learned and to what extent.
- Find better reflection questions to ask on the summary. Many times, in response to "What would you do differently," the student simply responded, "study more." While this may be true, it is not helpful for building better learning habits.
- Determine the best way to grade the summaries. We fell back to assigning the grade based on satisfactory completion of the homework and exercises. This brought us full circle, because the grade became practically the same as the traditional approach. Maybe this is OK, but we want to give the students some credit for the metacognition they are doing.
- Continue low-pressure individual meetings with students, as much as time allows.
- Investigate LMS support for alternative grading[2].
- Integrate AI and large language models in a way that is conducive to learning and student growth.

We instructors must be open to new ways to teach and assess. We used to assume that answers served as proof of work, and therefore proof of learning, with only occasional cases of copying that were easy to catch. In an information society where answers are readily available, we must continually remind

students that learning is not just answers and encourage them to push through the inherent challenges.

Furthermore, recognizing that these students will graduate soon and then enter a highly dynamic profession, we have the opportunity and obligation to force them to reflect on how they learn. This will differ from one student to the next, but with gentle prodding both by questions on the summaries and in meetings with the instructor, each student will come to understand how they best learn. This will prepare them for their future careers and for a lifetime of learning.

References

[1] David Clark and Robert Talbert. *Grading for Growth: A Guide to Alternative Grading Practices that Promote Authentic Learning and Student Engagement in Higher Education*. Routledge, 2023.

[2] Adrienne Decker et al. "A Call for Critical Technology to Enable Innovative and Alternative Grading Practices". In: *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1*. SIGCSE TS 2026. USA: Association for Computing Machinery, 2026, pp. 260–266. ISBN: 9798400722561. DOI: 10.1145/3770762.3772530.

[3] Joe Feldman. *Grading for Equity: What It Is, Why It Matters, and How It Can Transform Schools and Classrooms*. Corwin, 2018.

[4] Dan Garcia et al. "A's for All (As Time and Interest Allow)". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 1042–1048. ISBN: 9781450394314. DOI: 10.1145/3545945.3569847.

[5] Amruth N. Kumar et al. *Computer Science Curricula 2023*. New York, NY, USA: Association for Computing Machinery, 2024. ISBN: 9798400710339.

[6] Drew Lewis. "Impacts of Standards-Based Grading on Students' Mindset and Test Anxiety". In: *Journal of the Scholarship of Teaching and Learning* 22.2 (June 2022). DOI: 10.14434/josotl.v22i2.31308. URL: https://scholarworks.iu.edu/journals/index.php/josotl/article/view/31308.

[7] Scott Spurlock. "Improving Student Motivation by Ungrading". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 631–637. ISBN: 9781450394314. DOI: 10.1145/3545945.3569747.

[8] Robbie Weber. “Using Alternative Grading in a Non-Major Algorithms Course”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 638–644. ISBN: 9781450394314. DOI: 10.1145/3545945.3569765.

Benchmarking QAOA for Max-Cut and Resource-Constrained Project Scheduling: A Simulator and Hardware Performance Analysis*

Matteo Johnson, Anthony Gelasi, Hanwen Xing, Zulal Sevkli
Computer Science and Software Engineering
Miami University
Oxford, OH 45056
{john1798, gelasiae, xingh2, sevkliaz}@miamioh.edu

Abstract

Quantum approximate optimization is beginning to show measurable benefits on real quantum hardware for practical scheduling problems. This paper demonstrates one such application on IBM's 156-qubit "ibm_fez" backend. The Quantum Approximate Optimization Algorithm (QAOA) reduced project makespan, defined as the total time to complete all jobs in a schedule, by up to 21% compared to a classical solver on a 60-job resource-constrained scheduling benchmark, while matching or beating classical performance on five of six PSPLIB test instances overall. These findings are supported by a systematic evaluation of QAOA applied to the Max Cut problem on unweighted graphs with 4 to 24 nodes using an ideal Aer simulator. QAOA achieved perfect approximation ratios up to 12 nodes on a Max Cut problem and consistently outperformed classical brute-force in execution time as the problem size grew. Together, these experiments establish a progression from the foundational behavior of QAOA under ideal simulation conditions to its emerging practical potential on current noisy intermediate-scale quantum hardware.

1 Introduction

Combinatorial optimization problems represent a fundamental class of NP-hard challenges that arise across many real-world domains, including logistics, circuit design, finance modeling, network analysis, and project management. As problem sizes grow, the number of candidate solutions increases exponentially, making it computationally infeasible for classical algorithms to find optimal solutions within practical time limits. This scalability limitation has driven significant research interest into quantum computing as an alternative solution.

The Quantum Approximate Optimization Algorithm (QAOA), introduced by Farhi, Goldstone, and Gutmann [3], is a hybrid quantum-classical variational algorithm designed specifically for combinatorial optimization. QAOA encodes an optimization problem into a cost Hamiltonian and alternates between cost and mixer Hamiltonian layers to iteratively improve solution quality. Its shallow circuit depth makes it well suited for most noisy intermediate-scale quantum (NISQ) devices, which are constrained in qubit counts and coherence time.

This paper evaluates QAOA across two related but distinct problem settings. First, we benchmark QAOA against a classical brute-force solver on the Max-Cut problem using graphs from 4 to 24 nodes on a noise-free Aer simulator. Second, we embed QAOA as the Max-Cut oracle within a conflict-driven Resource-Constrained Project Scheduling (RCPS) solver, evaluating it on six PSPLIB benchmark instances on IBM's "ibmfez" hardware backend as well as on the simulator. Together, these two experiments provide a comprehensive empirical view of QAOA's practical strengths and limitations.

2 Problem Definition

2.1 Max-Cut Problem

The Max-Cut problem is a well-known NP-hard combinatorial optimization problem [7] with applications in circuit design, financial modeling, network analysis, and data flow optimization [4]. Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, the objective is to find a partition of V into two disjoint subsets A and B such that the number of edges crossing between the two subsets is maximized [3].

The Max-Cut problem is therefore defined as:

$$\max_{x \in \{-1,+1\}^n} \sum_{(i,j) \in E} \frac{1 - x_i x_j}{2} \tag{1}$$

2.2 Resource-Constrained Project Scheduling (RCPS)

Resource-Constrained Project Scheduling is a classical NP-hard combinatorial optimization problem [1] in which a set of jobs must be ordered and scheduled such that: (a) all precedence constraints between jobs are respected, and (b) no time-step exceeds the available capacity of any shared renewable resource [8]. A familiar version of this problem shows up on a construction site: framing a wall must happen before electrical wiring can run through it (a precedence constraint), and if only one crane is available, two crews cannot both use it at the same time (a resource constraint). RCPS arises in manufacturing, construction, software development, and project management [8]. The objective is to minimize the project makespan (the time at which the final job finishes), which is subject to both precedence and resource constraints.

3 QAOA Algorithm

QAOA is a hybrid quantum-classical variational algorithm first proposed by Farhi, Goldstone, and Gutmann for solving combinatorial optimization problems on near-term quantum devices [3]. Intuitively, QAOA puts every qubit into an equal mix of both possible states, then alternates two operations: a *cost* step, which nudges each connected pair of qubits toward disagreeing with each other (since disagreeing pairs are exactly the ones that count toward a larger cut), and a *mixer* step, which spreads probability back out so the circuit does not collapse onto the first promising partition it finds. Repeating this cost-then-mixer cycle p times gradually biases measurement outcomes toward stronger cuts. In this paper, a single layer ($p = 1$) is used, with fixed step sizes $\gamma = \pi/4$ and $\beta = \pi/2$, following [4]; rather than tuning these parameters iteratively, they are fixed in advance, trading some solution quality for a simpler, more reproducible circuit. Full mathematical derivations of the circuit, cost Hamiltonian, and mixer Hamiltonian are available in [3] and [10].

4 Proposed Algorithms

This section describes the proposed algorithms to solve the Max-Cut and RCPS problems.

4.1 Classical Brute-Force Solver for Max-Cut

The classical brute-force algorithm exhaustively evaluates all 2^n possible partitions of the vertex set V to find the one that maximizes the cut value $C(A, B)$. For each partition, represented as an n-bit integer, the algorithm iterates over all edges and counts the number of edges crossing between the two subsets.

The partition with the highest cut value is returned as the optimal solution. The implementation of the brute-force solver can be found in [6].

The time complexity of this algorithm is $O(2^n \cdot m)$, where n is the number of vertices and m is the number of edges.

4.2 QAOA for Solving Max-Cut

The framework proposed by Huang [4] was adopted to solve the Max-Cut problem using QAOA. For each unweighted graph $G = (V, E)$ with n vertices, an n-qubit QAOA circuit is constructed, where each qubit corresponds to a vertex in the graph. The measurement outcome of each qubit determines the partition (A or B) to which the corresponding vertex is assigned.

4.3 The Conflict-Resolution Algorithm for solving RCPS

The algorithm adopted from [2] starts with an iterative decomposition approach in which an Earliest Start Time (EST) scheduler first builds the most optimistic feasible schedule by ignoring resource constraints. Then a Max-Cut solver resolves resource violations by partitioning conflicting jobs. This process repeats until no conflicts remain or the iteration cap is reached.

The algorithm proceeds as follows:

1. **Schedule:** Compute the EST schedule via a topological forward pass, assigning each job the earliest start time that satisfies precedence and any start-floor constraints from prior iterations.

2. **Detect Conflicts:** Identify job pairs that overlap in time and jointly exceed a shared resource's capacity; these pairs form the edges of a conflict graph $G_c = (V_c, E_c)$.

3. **Solve Max-Cut:** Apply the classical solver or QAOA to partition V_c into two subsets, maximizing the number of conflict edges cut.

4. **Resolve Conflicts:** Push the smaller partition later in time by setting its start-floor to the larger partition's latest finish time:

$$\text{floor}(j) = \max_{k \in \text{larger}} \left(\text{start}(k) + \text{duration}(k)\right), \quad \forall j \in \text{smaller} \tag{2}$$

5. **Repeat:** Recompute the schedule with updated start floors; repeat until no conflicts remain or `MAX_ITER` $= 20$ is reached.

The objective of the algorithm is to minimize the project makespan - the time at which the final job completes - subject to both precedence and resource constraints. The makespan is formally defined as:

$$C_{\max} = \max_{j \in V} (\text{start}(j) + \text{duration}(j)) \tag{3}$$

The quality of the Max-Cut solution at each iteration directly influences the final makespan. A stronger cut resolves more conflicts per iteration by separating more conflicting job pairs into different partitions, potentially reducing the total number of iterations required and producing a shorter final schedule. A weaker cut may leave some conflicts unresolved, requiring additional iterations and pushing more jobs further out in time, resulting in a longer makespan.

The algorithm accepts any Max-Cut solver as a parameter, enabling a clean and fair comparison between the classical and quantum approaches within an identical scheduling framework.

5 Experimental Setting

5.1 Parameter Settings for QAOA

The QAOA implementation used in this paper follows the experimental configuration of Huang [4], using a fixed circuit depth of $p = 1$ with constant Hamiltonian parameters $\gamma = \pi/4$ and $\beta = \pi/2$. The circuit is constructed using IBM's Qiskit SDK [5] and executed on either a local Aer simulator or real IBM quantum hardware depending on the configuration toggle.

No classical optimizer is used in this work. The fixed parameter choice ensures consistency across all trials and graph sizes, making the experiment reproducible, at the cost of potentially lower approximation ratios than an optimized parameter search. The circuit is measured 2,048 times per trial. Each sampled bitstring is scored by counting edges whose endpoints fall in different partitions, and the bitstring with the highest cut value is retained as QAOA's proposed solution. Because measurement is stochastic, repeated runs on the same circuit may produce different partitions, which is the primary source of variance observed in the RCPS results.

5.2 Dataset

5.2.1 Graph Construction for Max-Cut

In this work, unweighted graphs are used exclusively, with all edge weights set to 1. Each graph is constructed as a polygon with n vertices arranged in a ring, where each vertex is connected to its two adjacent neighbors to form a closed cycle. Additional cross-edges are introduced for structural variety (for graphs with $n \geq 6$, an edge is added connecting node 0 to node $n/2$, and for graphs with $n \geq 8$, a second cross-edge connects node 1 to node $n/2 + 1$.)

5.2.2 PSPLIB Dataset

The RCPS experiments in this paper use benchmark instances drawn from the Project Scheduling Problem Library (PSPLIB), a widely used standard benchmark dataset for evaluating RCPS algorithms [8]. PSPLIB provides a large collection of problem instances of varying sizes and complexities, generated under controlled conditions to ensure reproducibility and comparability across solvers.

In this paper, six PSPLIB instances are evaluated: three 30-job instances from the `j30` benchmark set and three 60-job instances from the `j60` benchmark set. The 30-job instances produce smaller conflict graphs with fewer nodes and edges, making them more tractable for both the classical and quantum solvers. The 60-job instances produce larger conflict graphs, where the limitations of a fixed $p = 1$ QAOA circuit become more apparent in terms of both approximation quality and variance.

5.3 Performance Metrics

Three metrics are used to evaluate and compare algorithm performance.

- **Execution time:** Measured in seconds using Python's `time.perf_counter()` function. For the classical solver, the entirety of the brute-force function is timed. For QAOA, timing begins at circuit creation and ends when the job reaches completion.
- **Approximation ratio:** Calculated by dividing QAOA's best cut value by the optimal cut value returned by the brute-force solver. A ratio of 1.0 indicates that QAOA found the exact optimal solution. The brute-force solver always achieves a ratio of 1.0 by definition.
- **Circuit complexity:** Tracked by recording the qubit count, circuit depth, and total gate count for each graph size. This metric extends the analysis of Huang [4] by providing insight into how QAOA's computational overhead scales with problem size.

For each graph size, both algorithms are run for 5 independent trials and the average execution time and approximation ratio are reported.

6 Results

6.1 Max-Cut Results on Aer Simulator

Table 1 reports execution time and approximation ratio for both solvers across graph sizes 4 to 24. Classical execution time grows exponentially, consis-

tent with its $O(2^n)$ complexity, while QAOA's execution time remains near-constant, with the two approaches converging around 18–20 nodes.

Table 1: Max-Cut Results: Execution Time and Approximation Ratio

Nodes	Classical (s)	QAOA (s)	Approx. Ratio
4	0.000014	0.404	1.000
6	0.000138	0.275	1.000
8	0.001262	0.184	1.000
10	0.002328	0.150	0.967
12	0.012767	0.151	1.000
14	0.039896	0.159	0.875
16	0.186480	0.166	0.963
18	0.812507	0.216	0.830
20	4.113592	0.403	0.930
22	17.557616	1.265	0.817
24	75.055216	5.424	0.850

QAOA achieved a perfect approximation ratio of 1.0 on graphs up to 12 nodes, indicating that the $p = 1$ circuit with fixed parameters $\gamma = \pi/4$ and $\beta = \pi/2$ is sufficient to find the optimal partition on smaller problem instances. Beyond 12 nodes, the approximation ratio declined, reaching a minimum of 0.817 at 22 nodes before recovering slightly to 0.850 at 24 nodes. Circuit complexity scaled linearly with graph size, with qubit count growing from 4 to 24, circuit depth increasing from 15 to 78, and total gate count growing from 24 to 150.

6.2 RCPS Results on Aer Simulator

All RCPS simulator experiments were conducted on a noise-free local Aer simulator. Each instance was run five independent times and the average makespan and runtime are reported.

For the 30-job instances, the simulator QAOA matched or outperformed the classical method (Cl.) in two of three cases (Table 2). On the 60-job instances, it outperformed classical on all three, most notably on j60_2 with a 30.6% shorter makespan. Runtime was also dramatically lower on the larger instances — j601_2 completed in 9.3 seconds, compared to 507.1 seconds classically.

6.3 RCPS Results on IBM ibm_fez Hardware

All RCPS hardware experiments were conducted on IBM's `ibm_fez` 156-qubit quantum backend. Each instance was run 5 times, and the results were aver-

Table 2: RCPS Results — Aer Simulator (QAOA Averaged over 5 Runs)

Instance	Jobs	Makespan(Cl.)	Makespan(QAOA)	Time(Cl., s)	Time(QAOA, s)
j301_1	30	67	75	0.0	0.8
j301_2	30	76	74	0.1	0.4
j301_3	30	93	82	0.0	0.5
j601_1	60	191	164	63.1	1.9
j601_2	60	242	168	507.1	9.3
j601_3	60	128	122	2.3	0.7

aged to account for stochastic hardware variability (see Table 3).

Table 3: RCPS Results — IBM ibm_fez Hardware (QAOA Averaged over 5 Runs)

Instance	Jobs	Makespan(Cl.)	Makespan(QAOA)	Time(Cl., s)	Time(QAOA, s)
j301_1	30	67	75	0.0	41.7
j301_2	30	76	70	0.1	23.3
j301_3	30	93	70	0.0	28.4
j601_1	60	191	182	56.7	170.2
j601_2	60	242	212	465.4	124.9
j601_3	60	128	124	2.0	54.8

For the 30-job instances, hardware QAOA matched or outperformed classical methods in two of three cases, with j301_2 and j301_3 producing makespans of 70 versus 76 and 93, respectively. On the 60-job instances, hardware QAOA outperformed the classical method in all three cases, with j601_2 producing a makespan of 212 versus 242 classically. However, the hardware QAOA runtime was substantially higher than the simulator's due to physical circuit execution and backend queue time, with j601_1 averaging 170.2 seconds compared to 1.9 seconds on the simulator. The standard deviation of the hardware makespan results was also larger than that of the simulator across all instances. The sources of this variability are discussed in the next section.

7 Discussion and Conclusion

This paper presents a combined evaluation of QAOA across two NP-hard combinatorial optimization problems: the Max-Cut problem and the Resource-Constrained Project Scheduling problem. By replicating and extending the experimental framework of Huang [4] and embedding QAOA as a drop-in replacement for the classical Max-Cut oracle within a conflict-resolution schedul-

ing loop, this paper demonstrated both the strengths and current limitations of QAOA on near-term quantum devices.

The Max-Cut results confirm that QAOA's execution-time advantage over classical brute-force grows with problem size, even as its approximation ratio degrades on larger graphs under a fixed $p = 1$ circuit; a direct tradeoff between scalability and solution quality that motivates deeper circuits in future work. The RCPS results translate this scalability advantage into a practical setting: QAOA matched or outperformed the classical solver on five of six PSPLIB instances, with the largest gains on the 60-job instances, where classical brute-force's exponential cost becomes the dominant bottleneck.

Several limitations remain. Max-Cut benchmarks were run only on a noise-free simulator, while RCPS experiments also included IBM's `ibm_fez` hardware, which showed consistently higher variance and longer runtimes than the simulator. This difference is attributable to gate noise and decoherence, whose effects accumulate in the deeper circuits required for larger instances [9]. The fixed $p = 1$ circuit depth and constant Hamiltonian parameters used throughout this paper further limit solution quality on larger problem instances.

Hardware variance was concentrated in runtime, not solution quality. The coefficient of variation in runtime averaged 75% (30-job) and 43% (60-job), versus just 9% and 16% for makespan. The extreme case, j301_3, ranged from 7.3 to 95.6 seconds across five identical runs, representing a 13× difference in runtime. This variation was traced to the per-iteration cost, which ranged from 3.6–5.3 seconds in four runs but increased to 31.9 seconds in one run, suggesting that queue and dispatch latency on IBM Cloud, rather than quantum circuit execution, was the primary source of the runtime variability.

The standard deviation of makespan was comparable between hardware and the noise-free simulator at 30 jobs (6.1 vs. 10.2, respectively), and increased consistently with the number of conflict-resolution iterations a run required; on j601_2, makespans of 141 and 269 corresponded to five- and nine-iteration runs, respectively. This suggests dispersion stems mainly from the conflict-resolution loop's bitstring sampling, with gate error and decoherence a secondary factor at 60 jobs.

Future work should investigate the impact of increasing circuit depth beyond $p = 1$ and incorporating classical parameter optimization techniques such as Nelder-Mead or gradient descent to improve approximation quality. Finally, extending the RCPS experiments to larger PSPLIB instances from the `j90` and `j120` benchmark sets would provide further insight into how QAOA's scheduling performance scales with problem complexity.

All experiments in this paper are reproducible using Qiskit, IBM's open-source quantum computing SDK [5]; the full Max-Cut implementation is available as a Colab notebook [6].

References

[1] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. "Scheduling subject to resource constraints: Classification and complexity". In: *Discrete Applied Mathematics* 5.1 (1983), pp. 11–24. URL: https://www.sciencedirect.com/science/article/pii/0166218X83900124.

[2] Zsolt Csuka. "Solving project scheduling problems by maximum cut computation". In: *CompSysTech '10: Proc. 11th Int. Conf. on Computer Systems and Technologies*. Sofia, Bulgaria, June 2010, pp. 223–228. URL: https://dl.acm.org/doi/epdf/10.1145/1839379.1839419.

[3] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A quantum approximate optimization algorithm". In: *arXiv preprint arXiv:1411.4028* (2014). URL: https://arxiv.org/pdf/1411.4028.

[4] Yohhaan Yung Kang Huang. "Quantum approximate optimization algorithm for the Max-Cut problem: Performance comparison with classical approaches on NISQ devices". In: *Exploratio Journal* (Oct. 2025). URL: https://exploratiojournal.com/quantum-approximate-optimization-algorithm-for-the-max-cut-problem-performance-comparison-with-classical-approaches-on-nisq-devices/.

[5] IBM Quantum. *Qiskit: An open-source SDK for working with quantum computers at the level of extended quantum circuits, operators, and primitives*. 2023. URL: https://github.com/Qiskit/qiskit.

[6] Matteo Johnson, Anthony Gelasi, and Hanwen Xing. *QAOA Max-Cut Implementation*. Google Colaboratory. 2025. URL: https://colab.research.google.com/drive/1swgMjlIZEnY1NLkFDGqjlXrUfjv8TG2m?usp=sharing.

[7] Richard M Karp. "Reducibility among combinatorial problems". In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. New York: Plenum Press, 1972, pp. 85–103. URL: https://www.cs.umd.edu/~gasarch/BLOGPAPERS/Karp.pdf.

[8] Rainer Kolisch and Arno Sprecher. "PSPLIB — A project scheduling problem library". In: *European Journal of Operational Research* 96.1 (1997), pp. 205–216.

[9] John Preskill. "Quantum computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79. URL: https://quantum-journal.org/papers/q-2018-08-06-79/pdf/.

[10] Leo Zhou et al. "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices". In: *Phys. Rev. X* 10 (2020), p. 021067.

Using the Game Design Domain to Teach Software Specification*

Ashelyn Reilly and Sugandha Malviya
Department of Computer Science
Ball State University
Muncie, IN 47306
{ashelyn.reilly, sugandha.malviya}@bsu.edu

Abstract

This paper presents a lightweight approach for teaching software specification concepts using the game design domain. The approach analyzes gameplay descriptions from well-known games including *The Legend of Zelda: Breath of the Wild* and *Tetris*, and demonstrates how vague gameplay ideas can be refined into precise and testable software requirements.

The results show that gameplay descriptions provide an effective context for discussing important software specification concepts such as ambiguity, decomposition, precision, and validation. The analysis further demonstrates that gameplay signals, including music changes, visual indicators, checkpoints, and movement changes, can serve as an intermediate layer between informal gameplay ideas and observable system behavior. We further show how the resulting requirements can be validated through simple acceptance test cases.

The proposed approach highlights the educational potential of gameplay systems for teaching software specification concepts through familiar gameplay interaction and mechanics. Because gameplay systems involve visible player interactions, feedback and system responses, they may help students better connect software specification concepts to implementation and testing.

1 Introduction and Background

Teaching software specification concepts can be challenging because students often struggle to transform vague ideas into precise and testable system behavior [2]. Students may conceptually understand what a system should accomplish, but they frequently have difficulty describing the exact behaviors needed for implementation and testing. Concepts such as ambiguity, decomposition, precision, and validation are therefore often difficult to teach using traditional specification examples [1, 3].

The game design domain provides an engaging context for teaching these concepts [6]. Gameplay descriptions naturally communicate player experiences using informal and experience-oriented language such as *challenge*, *tension*, *exploration*, or *safety* [4]. These descriptions are intentionally high-level and often focus on how the player should feel rather than how the system should behave. For example, a gameplay description may state that "the player should feel pressure when enemies are nearby" without specifying the exact behaviors needed to create or validate that experience.

This ambiguity creates opportunities for students to practice refining informal gameplay ideas into more precise and testable system behavior. Additionally, games naturally contain observable mechanics, visual feedback, audio cues, environmental interactions, and player actions that can be mapped into executable system behavior.

In software engineering, software specifications describe expected system behavior in a precise and structured manner to support communication, implementation, testing, and validation [5]. Although software specifications may include both functional and non-functional aspects, this work focuses primarily on **functional requirements** that describe observable system behavior and interactions.

This work is further motivated by the relationship between gameplay descriptions, **experience goals**, **gameplay signals**, and functional software specifications. In this work, experience goals refer to the intended emotional or perceptual experiences of the player, such as tension, pressure, curiosity, or safety. Because these goals are often subjective, they can be difficult to refine into precise and testable functional requirements.

To address this challenge, we use gameplay signals which represent observable mechanics, feedback systems, or interactions that communicate information to the player. Examples include flashing health indicators, music changes, movement limitations, checkpoints, enemy behavior, or resource scarcity. These signals provide a more concrete foundation for translating gameplay experiences into observable system behavior.

In this paper, we present a lightweight approach for using the game design domain to teach software specification concepts. We analyze gameplay

descriptions from well-known games, extract experience goals, identify gameplay signals, and translate them into functional software specifications using a consistent requirement template. We further demonstrate how the resulting specifications can support validation through acceptance test cases.

The main contributions of this paper are as follows. First, we present a structured approach for refining informal gameplay descriptions into functional software specifications using experience goals and gameplay signals. Second, we illustrate how the game design domain can support the teaching of software specification concepts through concrete examples involving ambiguity, decomposition, precision, validation, and acceptance testing.

The remainder of the paper is organized as follows. Section 2 describes the proposed methodology. Section 3 presents the results and discussion. Section 4 highlights the educational perspective, and Section 5 concludes the paper and discuss future work.

2 Methodology

Our approach uses a lightweight, structured process for translating informal gameplay descriptions into executable functional requirements. The process consists of five stages: (1) collecting gameplay descriptions, (2) extracting experience goals, (3) identifying gameplay signals, (4) translating gameplay signals into functional requirements, and (5) validating selected requirements through acceptance test cases. Each stage represents a distinct refinement task that can be introduced independently or combined as part of a complete specification activity. Figure 1 provides an overview of this process.

Figure 1: Overview of the proposed gameplay refinement process

2.1 Gameplay Description Collection

Gameplay descriptions were collected from publicly available sources such as Wikipedia articles, community-maintained game wikis, gameplay summaries, and mechanics descriptions. These artifacts often described player experiences, gameplay mechanics, environmental interactions, rewards, penalties, and feedback systems using informal and experience-oriented language.

Many descriptions emphasized gameplay intent through terms such as *pressure*, *danger*, *exploration*, *speed*, or *safety* rather than explicit system behavior.

These descriptions served as the starting point for the specification refinement process.

2.2 Extraction of Experience Goals

The collected gameplay descriptions were manually analyzed to identify player-oriented experience goals communicated through gameplay interactions and mechanics. Examples included gameplay descriptions involving pressure from nearby enemies, safety through checkpoints or recovery systems, exploration through open environments, and urgency created by timers or resource limitations.

These experience goals helped capture the intended gameplay experience and served as the basis for identifying observable gameplay behaviors during later refinement stages.

2.3 Identification of Gameplay Signals

After identifying experience goals, we extracted associated gameplay signals from the gameplay descriptions. These signals included observable mechanics, feedback systems, and environmental responses such as flashing health indicators, music changes, stamina depletion, checkpoints, enemy pursuit behavior, movement restrictions, and environmental damage.

The extracted gameplay signals helped connect subjective gameplay experiences to more observable and testable system behavior, supporting the refinement of gameplay descriptions into functional requirements.

2.4 Translation into Functional Requirements

The identified gameplay signals were then translated into functional requirements describing observable system behavior. The requirements were written using the following structured template:

> *When [condition], the system shall [observable behavior].*

For example:

> *When the player's health falls below a defined threshold, the system shall display a flashing health indicator.*

Gameplay ideas containing multiple behaviors were decomposed into smaller functional requirements describing individual system behaviors, feedback mechanisms, or state changes to improve clarity, and testability.

2.5 Validation Through Acceptance Test Cases

Finally, selected specifications were validated through simple acceptance test cases written using observable conditions and expected outcomes. For example, a requirement involving low player health could be validated by reducing the player's health below a threshold and observing whether the flashing health indicator appeared.

3 Results

This section presents representative examples demonstrating how gameplay descriptions were refined into functional requirements and acceptance test cases using the proposed methodology. The examples focus on gameplay descriptions from *The Legend of Zelda: Breath of the Wild* and *Tetris*.

3.1 The Legend of Zelda: Breath of the Wild

The Legend of Zelda: Breath of the Wild is an open-world action-adventure game in which the player explores the world of Hyrule, completes quests and puzzles, regains lost memories, and attempts to defeat Calamity Ganon [8, 9]. The game emphasizes exploration, survival, combat, stamina management, and environmental interaction, making it a useful example for analyzing experience-oriented gameplay descriptions.

The analyzed gameplay descriptions revealed several experience goals related to alertness, danger, pressure, survival, and movement limitation. These goals were associated with gameplay signals such as music changes, flashing health indicators, enemy behavior, environmental damage, and stamina depletion. Table 1 presents representative examples of the extracted experience goals and gameplay signals.

Table 1: Example Experience Goals and Gameplay Signals

Experience Goal	**Gameplay Signal**
Alertness for monsters	Music plays when monsters are nearby
Dying	Health indicators and player character flash at critical health
Cold	Player character shivers and takes damage in cold environments
Pressure Continues	Blood Moon events revive defeated enemies
Slowing Down	Low stamina slows or stops movement

The gameplay signals were then translated into functional requirements describing observable system behavior. Representative examples included:

1. When the player enters the vicinity of monsters, the system shall play alert music.
2. When the player enters the vicinity of monsters, the system shall cause nearby monsters to chase the player.
3. When the player sustains critical damage, the system shall cause the health indicators to flash.
4. When the player sustains critical damage, the system shall cause the player character to flash red.
5. When the player enters a cold environment without protection, the system shall cause the player character to shiver.
6. When the player enters a cold environment without protection, the system shall gradually reduce the player's health.
7. When the player performs an action requiring stamina, the system shall decrease the stamina level.
8. When the player's stamina falls below a defined threshold, the system shall reduce the player's movement speed.
9. When the player's stamina is depleted, the system shall prevent the player from performing stamina-based actions.

The resulting functional requirements could then be connected directly to acceptance testing. For example, the requirement involving monster proximity and alert music was translated into the following acceptance test case.

These examples demonstrate that a single gameplay experience may require multiple functional requirements to fully describe the corresponding system behavior. For example, gameplay experiences related to danger and survival resulted in requirements involving music changes, health indicators, environmental effects, enemy behavior, and movement limitations. The decomposition process helped reduce ambiguity while improving precision and testability.

3.2 Tetris

We used *Tetris* as another example to analyze gameplay descriptions centered around timing, spatial awareness, and progression mechanics. *Tetris* is a puzzle game in which players arrange falling tetromino blocks to complete horizontal

Table 2: Example Acceptance Test Case

Requirement	When the player enters the vicinity of monsters, the system shall play alert music.
Precondition	The player is located in an area containing at least one monster, and no alert music is currently playing.
Test Steps	1. Start with the player outside the monster detection range. 2. Move the player into the defined vicinity of a monster. 3. Observe the background music.
Expected Result	The system plays alert music when the player enters the monster's vicinity.
Pass Criteria	The alert music begins after the player enters the defined monster detection range.
Fail Criteria	The alert music does not begin, begins before the player enters the range, or is triggered by an unrelated event.

lines while preventing the blocks from reaching the top of the play area [7]. As gameplay progresses, the falling speed of the blocks increases, requiring faster decision making, adaptability, and strategic placement.

The analyzed gameplay descriptions revealed experience goals related to pressure, problem-solving, spatial awareness, satisfaction, and strategic planning. These goals were associated with gameplay signals such as increasing block speed, block rotation, line clearing, stacking behavior, and previewing upcoming blocks. Table 3 presents representative examples of the extracted experience goals and gameplay signals.

The gameplay signals were then translated into the following functional requirements describing observable system behavior.

1. When the game begins, the system shall generate blocks using a predefined set of tetromino shapes.

2. When the player rotates a block, the system shall update the orientation of the block.

3. As gameplay progresses, the system shall increase the falling speed of the blocks.

Table 3: Example Experience Goals and Gameplay Signals in *Tetris*

Experience Goal	**Gameplay Signal**
Spatial Awareness	Blocks appear in limited shapes that fit together in specific ways
Pressure	Blocks begin falling faster as gameplay progresses
Problem-Solving	Blocks can be rotated to fit surrounding pieces
Satisfaction	Completed lines disappear from the play area
Adaptability	Blocks stack dynamically based on player placement
Strategic Planning	The system displays the next upcoming block

4. When the player completes a horizontal line, the system shall remove the completed line from the play area.
5. When a completed line is removed, the system shall shift the remaining blocks downward.
6. When the player is placing a block, the system shall display the next upcoming block.
7. When stacked blocks reach the top boundary of the play area, the system shall end the game.

The acceptance test case shown in Table 4 demonstrates how the requirement involving line clearing can be validated through observable gameplay behavior.

Unlike the *Breath of the Wild* examples, the *Tetris* requirements focused more heavily on timing, progression, spatial reasoning, and system-state changes rather than environmental interaction or survival mechanics. These examples demonstrate that the proposed approach can also be applied to abstract and mechanics-driven gameplay systems.

3.3 Analysis and Discussion

Several consistent patterns emerged across the analyzed gameplay descriptions. First, gameplay descriptions frequently emphasized intended player experiences such as pressure, danger, exploration, speed, or strategic planning rather than explicit system behavior. As a result, many gameplay descriptions required refinement before they could be translated into functional requirements.

Across both *The Legend of Zelda: Breath of the Wild* and *Tetris*, gameplay signals consistently acted as an effective intermediate layer between gameplay

Table 4: Example Acceptance Test Case for *Tetris*

Requirement	When the player completes a horizontal line, the system shall remove the completed line from the play area.
Precondition	The play area contains a nearly completed horizontal line with one remaining empty space.
Test Steps	1. Position a falling block above the remaining empty space. 2. Place the block to complete the horizontal line. 3. Observe the play area after the line is completed.
Expected Result	The completed horizontal line is removed from the play area.
Pass Criteria	The completed line disappears immediately after completion and the remaining blocks shift downward appropriately.
Fail Criteria	The completed line does not disappear, disappears incorrectly, or the remaining blocks do not shift properly.

descriptions and observable system behavior. Common gameplay signals included music changes, visual indicators, movement changes, timers, scoring systems, progression mechanics, and environmental effects. These signals helped transform subjective gameplay experiences into more precise and testable functional requirements.

The analysis also revealed differences in the types of requirements generated across genres. The *Breath of the Wild* examples focused more heavily on environmental interaction, survival mechanics, combat behavior, and resource management, while the *Tetris* examples emphasized timing, spatial reasoning, scoring, and progression mechanics. Despite these differences, the same refinement process could be applied across both games.

Another recurring observation was the role of decomposition in the specification process. Many gameplay experiences required multiple functional requirements to fully describe the intended behavior. For example, gameplay experiences related to danger or pressure often involved combinations of audio feedback, visual indicators, movement restrictions, environmental effects, or progression changes. Breaking these larger gameplay experiences into smaller requirements improved precision and supported validation through observable outcomes.

The resulting requirements also supported acceptance testing by expressing behavior using observable conditions and expected outcomes, reinforcing the relationship between specification, implementation, and validation.

While the proposed approach demonstrated useful educational potential, several considerations should be noted. The extraction of experience goals and gameplay signals was performed manually and may involve subjective interpretation. Additionally, the analyzed gameplay descriptions were collected from informal public sources rather than official game design documents. Nevertheless, the refinement process provided a useful framework for connecting informal gameplay ideas to functional requirements, observable system behavior, and acceptance testing.

4 Educational Perspective

The proposed approach may provide an engaging way to introduce software specification and requirements engineering concepts through familiar gameplay systems. Because gameplay interactions naturally involve visible feedback, environmental interaction, progression mechanics, timing constraints, and state changes, they provide concrete examples for discussing concepts such as ambiguity, decomposition, observability, precision, and validation. The game design domain may be particularly useful in classroom settings because many students are already familiar with gameplay interactions and can more easily reason about system behavior through recognizable examples.

The refinement process can support a sequence of short classroom activities, with each activity focusing on a different stage of software specification development. These activities can be used independently to introduce individual concepts or combined into a larger specification exercise. Early activities may focus on identifying ambiguity in gameplay descriptions and distinguishing subjective player experiences from observable system behavior. For example, students could analyze gameplay descriptions such as "the player should feel pressure when enemies are nearby" or "the player should feel urgency as the blocks begin falling faster" and discuss why these statements are difficult to implement or validate directly.

Additional activities may focus on identifying gameplay signals associated with a particular gameplay experience. Students could work individually or in groups to identify possible gameplay signals such as warning indicators, sound effects, movement restrictions, timers, enemy behavior, or environmental changes. Comparing alternative interpretations may encourage discussion around observability, interpretation, and requirement quality while illustrating how different gameplay experiences can be supported through multiple system behaviors.

To learn requirements decomposition, students could refine larger gameplay experiences into smaller functional requirements describing individual behaviors, feedback systems, or state changes. Students could further derive acceptance test cases from the resulting requirements by identifying observable conditions and expected outcomes. A peer-review activity could further ask students to evaluate the resulting requirements for clarity, observability, testability, completeness, and atomicity, and revise them based on feedback. These criteria could also form a lightweight rubric for instructor assessment or student self-evaluation.

The activities could be supported through reusable worksheets containing gameplay descriptions, requirement templates, assessment criteria, and example solutions. Together, these activities provide instructors with flexibility to introduce individual specification concepts through short exercises or combine the stages into a more complete requirements specification activity.

5 Conclusion and Future Work

This paper presented a lightweight approach for teaching software specification concepts using the game design domain. By analyzing gameplay descriptions from games such as *The Legend of Zelda: Breath of the Wild* and *Tetris*, the study demonstrated how informal and experience-oriented gameplay ideas can be systematically refined into functional requirements describing observable system behavior.

The results showed that gameplay signals can serve as an effective intermediate layer between gameplay descriptions and executable requirements. The refinement process also illustrated important software specification concepts such as ambiguity, decomposition, precision, and observability. The resulting functional requirements further demonstrated how observable system behavior can support testing through clear conditions and expected outcomes.

The game design domain provided a useful and accessible context for discussing software specification concepts because gameplay systems naturally involve visible feedback, progression mechanics, environmental interaction, timing constraints, and state changes that can be translated into observable and testable behavior. The familiarity of gameplay interactions may also help students better connect abstract software specification concepts to implementation, testing, and validation activities.

The current work focuses on the design and application of the proposed instructional approach and its effectiveness with students remains to be evaluated through classroom studies. Future work could examine how students interact with the proposed methodology and whether gameplay-based specification activities improve student understanding, engagement, or requirement

quality. Student-generated requirements could be assessed using the proposed quality criteria, while pre- and post-activity measures could examine changes in students' understanding of software specification concepts. A comparative study could further examine differences in learning and engagement between gameplay-based activities and more traditional software specification exercises. Additional future directions include automated extraction techniques and the use of large language models to support gameplay description analysis and requirement generation.

References

[1] Daniel M Berry and Erik Kamsties. "Ambiguity in requirements specification". In: *Perspectives on software requirements*. Springer, 2004, pp. 7–44.

[2] Anthea Moravánszky. "Challenges of Requirements Engineering Education in Higher Education: Insights from an Interview Study with Educators in Switzerland." In: *REFSQ Workshops*. 2025.

[3] Franklin Parrales-Bravo et al. "Dear: Detecting ambiguous requirements as a way to develop skills in requirement specifications". In: *Electronics* 13.15 (2024), p. 3079.

[4] Jesse Schell. *Tenth anniversary: The art of game design: A book of lenses*. AK Peters/CRC Press, 2019.

[5] Ian Sommerville. *Software Engineering*. 10th. Pearson, 2016. ISBN: 978-1-292-09613-1. URL: https://www.pearson.com/us/higher-education/program/PGM35255.html.

[6] Mei Teng Soo and Hazleen Aris. "Game-Based Learning in Requirements Engineering: An Overview". In: *2018 IEEE Conference on e-Learning, e-Management and e-Services (IC3e)*. 2018, pp. 46–51. DOI: 10.1109/IC3e.2018.8632650.

[7] Wikipedia Contributors. *Tetris*. 2025. URL: https://en.wikipedia.org/wiki/Tetris.

[8] Wikipedia Contributors. *The Legend of Zelda: Breath of the Wild*. 2026. URL: https://en.wikipedia.org/wiki/The_Legend_of_Zelda:_Breath_of_the_Wild.

[9] Zelda Wiki Contributors. *Enemies in Breath of the Wild*. 2026. URL: https://zeldawiki.wiki/wiki/Enemies_in_Breath_of_the_Wild.

The Oft-Overlooked Role in Software Engineering: A Position Paper on Teaching Business Analysis in Higher Education*

Tanner L. Little and David L. Largent
Department of Computer Science
Ball State University
Muncie, IN 47306
tanner.l.little@gmail.com, dllargent@bsu.edu

Abstract

Most software fails not because of code. It fails because people misunderstand one another.

Modern software systems are often described through the lens of engineering: code, architecture, infrastructure, and performance. In practice, the success or failure of these systems frequently depends on something else in addition—people's ability to interpret, communicate, and align their understanding of what software is intended to accomplish.

Despite this reality, the disciplines responsible for building software and those responsible for leveraging it and operating organizations are traditionally taught in isolation. Computer science programs emphasize technical development, while business education focuses on strategy, operations, and management. Between these two domains lies a critical but often overlooked function.

Enter the Business Analyst.

While some interdisciplinary programs touch on aspects of technical and organizational integration, there remains no formal, consistently

structured training that prepares students to fulfill a Business Analyst role, particularly the ability to translate between technical constraints and organizational intent. Business Analysts operate within a socio-technical environment in which software systems must simultaneously reflect technical constraints, business objectives, and human workflows. This role often serves as a bridge between groups with different expertise, perspectives, and incentives—translating organizational intent into technical specifications and technical constraints into operational reality.

Despite its importance, the discipline remains underrepresented in undergraduate education. Computer science programs emphasize the mechanics of building systems, while business programs—even those with information systems curricula—focus on organizational strategy and operations. The work required to connect these domains—requirements discovery, stakeholder alignment, and system interpretation—is rarely treated as a core academic objective.

We examine the "human layer" of software engineering and argue that many project failures stem not from incorrect code but from misaligned understanding among the individuals responsible for developing, defining, and delivering systems, highlighting the need for a formal curriculum designed to bridge this gap.

1 Introduction

While software systems are built with code, their success or failure is frequently determined by the quality of human communication surrounding them.

Modern software systems are often described through the lens of engineering: code, architecture, infrastructure, and performance [1]. When software projects struggle, attention commonly turns to technical origins, items such as poor tooling, faulty algorithms, unstable infrastructure, or inadequate system design [3]. From this perspective, success or failure appears to depend primarily on engineering capability. Yet practitioners working within software organizations frequently observe a different reality.

While software can fail due to technical issues, arguably more projects falter primarily because of misalignment, miscommunication, and misunderstanding among the people responsible for designing, building, and using systems. These breakdowns are often influenced by gaps in formal education, which rarely prepares students to navigate the socio-technical complexities of real-world software development. Our use of the term socio-technical refers to scenarios where technical implementation must operate in constant coordination with the larger constructs of professional society, be it communication, organizational dynamics, stakeholder expectations, or business needs [2].

Within this context, the Business Analyst plays a critical role: translating organizational intent into technical specifications, reconciling differing stake-

holder perspectives, and ensuring that systems function not only correctly but meaningfully within their intended environment [7]. Although one piece of a larger modernization process, the work of the Business Analyst is integral to bridge the divide between technical implementation and organizational needs.

1.1 The Authors' Perspectives

This paper was inspired by the experiences of the two authors, bringing together practitioner and academic perspectives. Tanner Little is a practicing Senior Business Analyst and Project Manager who has witnessed firsthand how systems that function technically can still fail to deliver real organizational value when alignment and processes are lacking. David Largent is a Senior Lecturer of Computer Science who entered academia following decades of industry experience. He regularly sees learners who are highly focused on coding and building systems but who often struggle to recognize the value of developing non-domain workforce competencies identified by NACE, such as communication [11]. Together, these perspectives offer a balanced lens through which both the practical and educational dimensions of Business Analysis are evaluated[1].

2 The Current State

Software development at the pace and scale of today's world is rarely a solitary activity [6]. Instead, it is a collaborative process involving a wide range of participants, including developers, architects, project managers, policy experts, business leaders, domain specialists, subject matter experts, end users, and a host of other disciplines. Each of these groups approaches systems from a different perspective, guided by distinct expertise, vocabulary, priorities, culture, and other intangibles. To paint with broad brushes: developers reason about data structures, logic, and system constraints while business leaders think in terms of operational outcomes and organizational strategy—and ultimately, end users evaluate systems according to how effectively and easily the systems meet their needs.

The challenge in software development, therefore, extends beyond writing code. It involves translating organizational goals into technical systems that behave predictably within real-world environments and satisfy human needs. Misunderstandings can occur at any point within this process. Prototypical examples include stakeholders who describe their needs in ambiguous language or developers who may interpret those needs through technical assumptions

[1] While earning a Bachelor's degree from Ball State University, Tanner was one of David's students.

that differ from stakeholder expectations. Over time, these small discrepancies can accumulate, producing systems that technically function, yet fail to meet organizational or human needs.

2.1 Socio-technical Systems

Recognizing this dynamic requires viewing software development not solely as an engineering discipline but as a socio-technical process. Software systems do not exist independently of the environment in which they operate. They are embedded within organizations, policies, workflows, and human behaviors. The effectiveness of a software system therefore depends not only on its technical implementation but also on how well it aligns with the people and processes surrounding it. After all, software should serve people as a helpful extension of the human experience, not the other way around.

Within socio-technical systems, communication and interpretation play a central role. This occurs because most typically, real-world development does not occur within a vacuum. Stakeholders must articulate goals that are often incomplete or evolving. Development teams must interpret those goals while navigating technical constraints. Meanwhile, organizational priorities may shift as projects progress, requiring continual adjustment and clarification. These interactions form the foundation of software development, yet they are rarely treated as a central subject of study within computer science, associated technology, or business education.

Communication challenges within software projects are widely recognized by practitioners and academia, alike [13]. Requirements are often expressed in general or ambiguous terms that lack the specificity needed for accurate technical implementation, making them unclear, non-actionable, or difficult to measure. Stakeholders may assume that their meaning is obvious, even though interpretations differ. Developers, in turn, may rely on prior experience or assumptions rather than explicit confirmation. As a result, even well-managed teams can find themselves in situations where participants believe that they are discussing the same concept, yet each is referencing a different interpretation, creating subtle misalignments that can ripple through a project. While some failures are purely technical, many technical issues—such as performance problems, defects, or misbehavior—can originate from these misalignments and breakdowns in communication and process execution.

Such misunderstandings are not necessarily the result of negligence or incompetence. Instead, they arise from the inherent complexity of translating ideas across domains. Organizational leaders speak in the language of policy, outcomes, constraints—engineers reason about architectures, interfaces, system behavior—and end users evaluate software through the lens of everyday work practices. And rightfully so, since we, as people, tend to approach things

from a position of familiarity. This is not negligence, it is convenience. Whereas aligning these perspectives requires deliberate effort.

3 The Emerging Role of the Business Analyst

Within professional software development environments, a role has emerged to address this inconvenience or challenge: the Business Analyst [7]. Business Analysts operate within a socio-technical environment where software systems must simultaneously reflect technical constraints, business objectives, and human workflows. In other words, rather than focusing exclusively on system implementation or technical correctness, they focus on understanding and translating intent in an effort to achieve the aforementioned.

The work of a Business Analyst often begins with requirements discovery. Through interviews, workshops, document analysis, and observation of operational practices, analysts seek to understand the goals and constraints that a system must address. This process frequently reveals that the stakeholders themselves may hold different interpretations of the problem being solved. Clarifying these perspectives becomes an essential step before technical development can proceed effectively.

Once requirements are discovered, Business Analysts help translate them into forms that development teams can accurately interpret accurately. This translation involves more than simply documenting features. Analysts must identify the underlying assumptions, clarify ambiguous language, and reconcile conflicting priorities. In practice, this often means facilitating difficult conversations between stakeholders with competing expectations, uncovering gaps that were previously unnoticed, and refining requirements repeatedly as technical constraints and business realities become clearer. The resulting specifications represent a negotiated understanding between stakeholders and technical teams.

Business Analysts also play a role in validating systems once development begins. As software evolves, analysts help stakeholders verify that the system's behavior aligns with their expectations. When discrepancies arise, analysts facilitate discussions that lead to revised requirements or design adjustments. In this way, the analyst helps maintain alignment between organizational intent and technical implementation throughout the lifecycle of a project.

Despite the importance of this work, the discipline remains underrepresented in undergraduate education. Computer science programs emphasize the mechanics of building systems with subjects that are essential for producing technically capable engineers, yet they provide limited guidance on how to interpret organizational needs or manage communication across stakeholder groups [10].

Business schools approach the problem from a different vantage point. Their curricula typically emphasize management, organizational behavior, finance, and strategic decision-making [5]. While these subjects address the broader context in which organizations operate, they rarely examine how complex software systems are specified, interpreted, and validated during development.

The skills associated with Business Analysis reflect a broader dimension of human experience that extends beyond formal education and professional training [7]. At its core, this work involves interpreting intent, facilitating shared understanding, and reconciling differing perspectives within complex environments. Effective communication, social awareness, and emotional understanding are central to this process, shaping not only the quality of interactions but also the ability of individuals to operate as effective practitioners. While these capabilities are often framed in terms of their impact on software delivery, they are equally relevant to how individuals communicate, collaborate, and navigate ambiguity in a wider context. In this regard, Business Analysis is not solely a function of building better systems, but of fostering more effective interaction within the environments people inhabit.

A typical computer science curriculum focuses on algorithms, data structures, operating systems, software architecture, computing theories, etc. [10]. Often, it will include a one- or two-semester senior capstone course during which small teams of students strive to develop a solution for an external client. They may experience some ambiguity and misinterpretation during such a course, but the solutions they develop are typically small, compared to what they will experience in industry, and thus may provide only minimal insight into what they will experience later. David has included coursework related to the NACE workforce competencies, when possible, especially those of communication, equity and inclusion, leadership, and teamwork. However, without repeated emphasis throughout the rest of the curriculum, these efforts may not have a significant impact on the graduate.

Some Information Systems programs attempt to bridge this gap. However, these programs often emphasize enterprise system management, Information Technology governance, or organizational technology strategy rather than the detailed interpretive processes that occur within software development teams [5]. Coursework in these programs frequently focuses on how organizations select, manage, and maintain large technology systems, as well as how technology investments align with broader business objectives. While these perspectives are valuable for understanding the strategic role of technology within organizations, they tend to operate at a level of abstraction removed from the day-to-day interactions that shape software development itself.

Within development teams, the challenge is rarely limited to choosing the correct system architecture or aligning technology investments with organi-

zational strategy. Instead, it involves interpreting incomplete or evolving requirements, reconciling differing stakeholder perspectives, and translating operational needs into precise technical specifications that can be implemented by engineering teams. These activities require structured methods for discovery, clarification, and communication—skills that are central to professional Business Analysis but are rarely treated as a primary subject of study within Information Systems curricula. Information Systems programs provide valuable insight into how organizations govern and manage technology, but often leave unexamined the interpretive work that occurs at the intersection of human understanding and technical implementation during the development of software systems.

It is worth noting that elements of Business Analysis are addressed within certain advanced academic programs. Specialized graduate study in areas such as Human–Computer Interaction, information systems strategy, or MBA concentrations focused on technology management often explore topics related to user needs, system design, and organizational alignment. These programs can provide valuable insight into the human and organizational dimensions of technology development. However, positioning this type of work primarily at the graduate level risks overlooking its fundamental relevance to the broader practice of software development.

The interpretive and communicative processes that characterize Business Analysis are not advanced specializations reserved only for experienced practitioners; they are foundational competencies that shape the success of software systems from the earliest stages of development. As such, these concepts should not be treated solely as graduate-level topics but recognized as essential components of undergraduate education for those entering software-related fields.

As a result, many professionals who eventually work as Business Analysts enter the field through indirect pathways. Some arrive from technical backgrounds, discovering through experience that communication and requirements interpretation are central to project success. Others transition from operational roles, bringing domain expertise that allows them to interpret organizational needs for technical teams. Still others develop these skills through consulting work or project management experience. However, in most cases, formal academic preparation for this role remains limited.

The absence of structured education in Business Analysis has practical consequences. Development teams may lack individuals specifically trained to manage requirements discovery and stakeholder alignment. Developers are expected to interpret organizational needs directly, even when those needs are only partially articulated. Organizations may underestimate the complexity of translating policy or operational objectives into software functionality.

These conditions increase the likelihood of misalignment during develop-

ment. Teams may produce systems that technically satisfy documented requirements yet fail to meet stakeholder expectations. Alternatively, projects undergo repeated revisions as misunderstandings become apparent late in the development process, leading to material impact, including repeated revisions, delays, and additional cost. In many cases, the underlying issue is not faulty engineering, but incomplete or inconsistent understanding of the problem being solved.

4 A Professional Response to the Challenge

Recognizing this dynamic suggests that the socio-technical dimension of software development deserves greater attention within computer science education. While engineering expertise remains essential and paramount, technical skills alone do not ensure successful system outcomes. Students preparing to work in software development environments will benefit from exposure to the processes through which requirements emerge, evolve, and are interpreted.

Integrating Business Analysis concepts into computer science curricula could help address this gap. Such integration might include instruction in requirements elicitation methods, stakeholder analysis, collaborative decision-making, and techniques to validate system behavior against organizational goals. These topics would not replace traditional engineering coursework but would complement it by introducing structured approaches to interpreting, communicating, and refining requirements within real-world development contexts.

The objective is not to transform every Computer Science student into a Business Analyst, nor should undergraduate programs attempt to replace the experiential learning that naturally occurs in industry. Rather, the goal should be to establish a stronger baseline exposure to the socio-technical realities that surround modern software development. Students would benefit from greater familiarity with how requirements are elicited, interpreted, challenged, refined, and validated throughout the lifecycle of a project. This includes learning how to ask clarifying questions, identifying hidden assumptions, recognizing ambiguity within requirements, and distinguishing between what stakeholders say they want and the underlying organizational need actually being addressed.

Additional exposure in curriculum to communication and interpretive practices would also better prepare graduates for the modern workforce. This may include communicating technical concepts to non-technical stakeholders, facilitating conversations between groups with competing priorities, documenting requirements in structured and measurable ways, evaluating whether proposed solutions genuinely satisfy business objectives, and understanding how organizational processes shape system behavior. Students would similarly benefit from exposure to iterative refinement processes, stakeholder workshops, ven-

dor evaluations, and the broader challenge of translating human intent into technical implementation. Importantly, these concepts do not require universities to recreate entirely new educational models. Many of these competencies are already reflected within established professional frameworks such as the Project Management Institute PMP and PMI-PBA disciplines, which emphasize structured communication, stakeholder engagement, requirements management, validation, and organizational alignment as essential components of successful project delivery [8, 9].

Viewing software engineering through this broader lens highlights the existence of what might be described as the human layer of software systems. Beneath the technical architecture of a system lies a network of conversations, interpretations, and decisions that determine what the system ultimately becomes. Requirements are negotiated, priorities are balanced, and assumptions are clarified through human interaction long before code is written. This responsibility extends beyond requirements gathering alone. In practice, Business Analysts may also evaluate vendor proposals and solution designs, identifying when responses are overly generalized, lack measurable criteria, or fail to fully address organizational needs. In doing so, analysts help ensure not only that requirements are understood, but that proposed solutions can be reasonably validated against them.

Just as technologies evolve over time, the social environments in which they operate evolve as well. Organizations change, workflows adapt, and expectations about how technology should support human activity continue to shift. Software development therefore exists within a continually changing landscape in which both technical capabilities and social needs advance together. Over the past several decades, development practices have moved from tightly controlled procedural programming toward increasingly abstract forms of system creation, including distributed architectures, automated tooling, and low-code development environments [4]. More recently, advances in artificial intelligence (AI) have begun to transform how software artifacts themselves are produced. Systems capable of generating code, documentation, and system specifications from natural language inputs are rapidly becoming integrated into development workflows [12]. While these technologies promise to accelerate development and reduce manual effort, they also intensify the importance of understanding what systems are actually intended to accomplish.

The importance of human understanding within software development becomes even more pronounced in the context of artificial intelligence. Traditional software development requires engineers to translate human intent into explicit instructions that machines can execute. While this process is imperfect, the act of writing code forces developers to confront the assumptions hidden within a system's design. Requirements must be interpreted, constraints must be con-

sidered, and developers must continually evaluate whether the system they are building reflects the needs it is meant to serve.

Artificial intelligence systems alter this dynamic in important ways. Modern AI development increasingly relies on models that generate output based on patterns learned from data rather than explicit procedural instructions written by developers. In many cases, engineers are no longer defining every step of system behavior directly. Instead, they are shaping inputs, prompts, training data, and evaluation mechanisms that influence how models behave. As direct human authorship of system behavior becomes more abstract, the role of human interpretation becomes even more critical.

This shift changes how requirements must be interpreted and validated. Traditional software systems are frequently evaluated against precise and deterministic expectations, while AI systems may instead operate within ranges of acceptable outcomes. Determining whether a response is "correct" can depend on interpretation, context, and tolerance for variation rather than strict binary evaluation alone. Consequently, requirements for AI systems cannot always be fully defined upfront and instead require ongoing refinement, validation, and cross-functional review as systems are tested and used in practice.

These realities reinforce the broader socio-technical argument underlying this work. Ignoring this layer can lead to situations in which technically sophisticated systems fail to achieve their intended outcomes. Conversely, recognizing and addressing the human layer can improve collaboration, reduce misunderstandings, and increase the likelihood that systems deliver meaningful value.

5 Conclusion

Ultimately, improving software development requires more than better tools or programming techniques. It requires improving the ways people communicate, interpret, and align their understanding of complex systems. The role of the Business Analyst represents one professional response to this challenge.

By formally recognizing this discipline and incorporating its principles into computer science and associated education, institutions may better prepare students for the collaborative realities of modern software development. This approach acknowledges that software systems are not merely collections of code but expressions of shared understanding among the people who design, build, and use them.

When shared understanding is absent, software projects struggle regardless of technical sophistication. Yet alignment does not emerge organically; it is achieved through structured processes, disciplined methodologies, and deliberate facilitation between stakeholders with differing perspectives. The role of the

Business Analyst represents one professional response to this challenge, combining communication-oriented skills with formal practices such as requirements elicitation, stakeholder analysis, validation, and iterative refinement. When these processes are underdeveloped, misalignment persists regardless of technical capability. When they are applied effectively, even highly complex systems can deliver meaningful value. The future of software engineering, therefore, may depend as much on strengthening socio-technical practices as it does on advancing technical capability.

References

[1] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice, 4th Edition*. Addison-Wesley Professional, Aug. 2021. ISBN: 978-0136886099.

[2] Gordon Baxter and Ian Sommerville. "Socio-technical systems: From design methods to systems engineering". In: *Interacting with computers* 23.1 (2011), pp. 4–17. DOI: 10.1016/j.intcom.2010.07.003.

[3] Frederick P. Brooks. *The mythical man-month (anniversary ed.)* USA: Addison-Wesley Longman Publishing Co., Inc., Aug. 1995. ISBN: 978-0201835953.

[4] Brian Fitzgerald and Klaas-Jan Stol. "Continuous software engineering and beyond: trends and challenges". In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. RCoSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 1–9. ISBN: 978-1450328562. URL: https://doi.org/10.1145/2593812.2593813.

[5] The Joint ACM/AIS IS2020 Task Force, Paul Leidig, and Hannu Salmela. *A Competency Model for Undergraduate Programs in Information Systems*. New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 978-1450384643.

[6] GitHub. *Octoverse: AI leads Python to top language as the number of global developers surges*. 2024. URL: https://github.blog/news-insights/octoverse/octoverse-2024/.

[7] IIBA. *What is Business Analysis?* 2026. URL: https://www.iiba.org/professional-development/career-centre/what-is-business-analysis/.

[8] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Eigth Edition and The Standard for Project Management*. PMBOK® Guide. Project Management Institute, 2026. ISBN: 978-1628258295.

[9] Project Management Institute. *The PMI Guide to Business Analysis*. PMI global standard. Project Management Institute, 2018. ISBN: 978-1628251982.

[10] Amruth N. Kumar and Rajendra K. Raj. "Computer Science Curricula 2023 (CS2023): The Final Report". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2024. Portland, OR, USA: Association for Computing Machinery, 2024, pp. 1867–1868. ISBN: 979-8400704246. URL: https://doi.org/10.1145/3626253.3633405.

[11] NACE. *What is career readiness?* 2026. URL: https://www.naceweb.org/career-readiness/competencies/career-readiness-defined/.

[12] Erik Nijkamp et al. "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=iaYcJKpY2B_.

[13] Margaret-Anne Storey et al. "How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development". In: *IEEE Trans. Softw. Eng.* 43.2 (Feb. 2017), pp. 185–204. ISSN: 0098-5589. URL: https://doi.org/10.1109/TSE.2016.2584053.

Using Cisco Academy Platform to Teach Intro to Networks Course for CS Students *

Imad Al Saeed
Computer Sciences Department
Saint Xavier University
Chicago, IL 60655
alsaeed@sxu.edu

Abstract

Computer networking is a foundational subject in computer science curricula because it introduces students to network architecture, protocols, routing, switching, cybersecurity, and communication systems. Traditional approaches to teaching introductory networking courses often rely heavily on lectures and theoretical explanations, which may not provide sufficient hands-on experience for students. The Cisco Academy (NetAcad) platform has emerged as a widely adopted educational environment that combines online instructional content, simulation tools, assessments, and laboratory activities [3]. This research paper investigates the effectiveness of Cisco Networking Academy in teaching introductory networking courses to computer science students in the United States. The study includes qualitative data collection from professors teaching networking courses using Cisco Academy platforms. Findings indicate that instructors value the platform's hands-on learning environment, structured curriculum, and industry alignment. However, participants also identified challenges such as curriculum rigidity, certification-focused content, and the need for supplemental materials. The paper concludes that Cisco Academy significantly enhances networking education when integrated with active learning strategies and instructor customization.

1 Introduction

Computer networking courses are fundamental components of computer science and information technology curricula because they provide students with the knowledge and skills necessary to understand modern communication systems. Students pursuing computer science degrees must develop a solid understanding of network communication, routing, switching, wireless networking, cybersecurity, and Internet protocols. However, these topics can be challenging for beginners because many networking processes are abstract and occur behind the scenes, making them difficult to visualize and comprehend through theoretical instruction alone.

Traditional lecture-based teaching methods often provide limited opportunities for students to apply theoretical concepts in practical settings. Consequently, many colleges and universities have shifted toward laboratory-oriented instructional approaches that emphasize hands-on learning and simulation-based activities. These experiential learning environments allow students to develop practical networking skills while reinforcing classroom concepts. One of the most widely adopted educational platforms supporting this approach is the Cisco Networking Academy (NetAcad) [3].

Cisco Networking Academy was established in 1997 to address the growing demand for skilled networking professionals and support technical workforce development. Since its inception, the program has expanded into one of the world's largest technology education initiatives, serving more than 28 million learners globally [2] [3]. The Academy offers a comprehensive portfolio of courses in networking, cybersecurity, programming, the Internet of Things (IoT), and other computing-related disciplines. It also provides structured learning pathways aligned with industry-recognized certifications, including Cisco Certified Network Associate (CCNA), CyberOps Associate, Networking Essentials, and Introduction to Cybersecurity. These certification-focused pathways help students acquire both theoretical knowledge and practical skills that are directly applicable to careers in networking and information technology.

The Cisco Networking Academy learning environment integrates cloud-based curriculum modules, interactive multimedia lessons, online assessments, virtual laboratories, Cisco Packet Tracer simulations, instructor management tools, and certification-aligned training to create a comprehensive educational experience [3]. This blended learning approach allows students to study networking concepts independently while reinforcing their understanding through practical laboratory activities and instructor-guided instruction. By combining theoretical content with experiential learning, the platform supports the development of both conceptual understanding and technical competency.

One of the most valuable features of Cisco Networking Academy is Cisco

Packet Tracer, a powerful network simulation environment that enables students to design, configure, test, and troubleshoot virtual computer networks without requiring expensive physical networking equipment. Packet Tracer provides a realistic environment in which students can experiment with routers, switches, wireless devices, and network services while observing network behavior in real time. Research has demonstrated that Packet Tracer enhances students' understanding of networking concepts, improves problem-solving abilities, and increases engagement during laboratory activities by allowing learners to safely practice network configuration and troubleshooting in a simulated environment [8].

This paper examines the role of Cisco Networking Academy in supporting the instruction of introductory networking courses for computer science students at higher education institutions in the United States. In addition to evaluating the educational features of the platform, the study investigates instructors' experiences and perceptions through qualitative data collection methods to better understand the strengths, challenges, and overall effectiveness of Cisco Networking Academy in networking education.

2 Literature Review

Networking education has evolved significantly over the past two decades due to technological advancements and workforce demands. Universities increasingly emphasize experiential learning and practical laboratory experiences in computer science programs. Research shows that hands-on activities improve student engagement and technical competency in networking courses [5]. Students learn more effectively when theoretical concepts are reinforced through practical exercises. However, networking laboratories can be expensive to maintain because routers, switches, firewalls, and wireless equipment require substantial institutional investment. Virtual simulation platforms have therefore become attractive alternatives.

2.1 Cisco Packet Tracer and Simulation-Based Learning

Cisco Packet Tracer is among the most widely used networking simulation tools in education. It enables students to configure routers, switches, VLANs, routing protocols, and network services in a virtual environment. A recent study evaluating Packet Tracer found that the tool improves student achievement and attitudes toward networking education [8]. The study reported that collaborative and simulation-based learning approaches increased student understanding of networking concepts. Packet Tracer also supports collaborative learning environments that allow instructors to evaluate student configurations and troubleshooting skills remotely [6]. Students frequently describe Packet

Tracer as helpful for visualizing complex networking concepts and improving confidence during laboratory exercises.

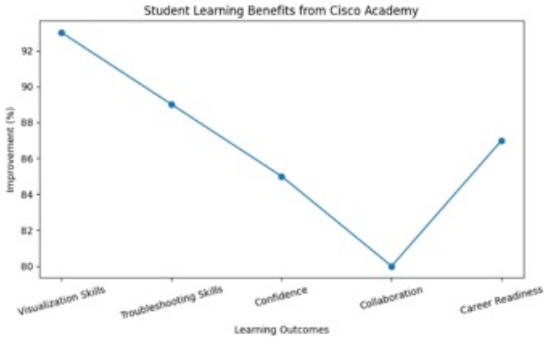


Figure 1: Students learning benefits from Cisco Academy

2.2 Blended Learning in Networking Education

Cisco Networking Academy uses a blended learning approach that combines online instruction with instructor-led classroom activities. Research examining the Cisco Academy model found that blended learning supports flexible and scalable networking education [7]. Blended learning environments allow students to:

- Review course materials independently
- Practice labs repeatedly
- Receive immediate assessment feedback
- Engage in collaborative exercises

Cisco Academy also provides standardized instructional materials that help instructors maintain curriculum consistency across institutions.

2.3 Criticism and Challenges of Cisco Academy

Although Cisco Academy offers many advantages, instructors and students have identified several limitations. Some instructors report that the curriculum can be overly certification-focused rather than concept-focused. Others argue that students sometimes memorize commands without fully understanding networking theory. As a result, supplemental teaching materials are often necessary. Additional challenges include:

- Steep learning curves for beginners
- Dependence on Cisco terminology
- Time-intensive laboratory activities
- Difficulty balancing theory and certification preparation

Despite these concerns, most educators acknowledge the value of practical networking experiences.

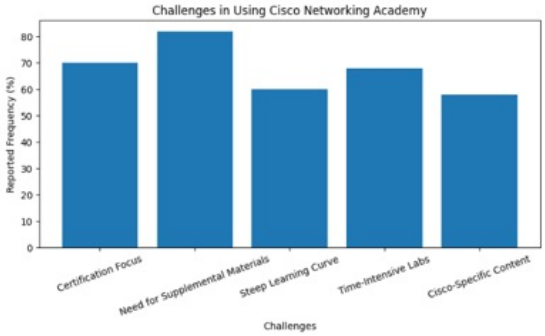


Figure 2: Challenging in using Cisco Academy

3 Problem Statement

Teaching introductory networking courses presents several educational challenges that can affect student learning and engagement. One of the primary difficulties is that many networking concepts, such as routing, switching, packet transmission, and protocol interactions, are highly abstract and difficult for students to visualize. In addition, many colleges and universities are unable to afford extensive networking hardware laboratories, limiting students' opportunities for hands-on experience with physical networking equipment.

Students also frequently struggle to understand packet flow and the operation of network protocols because these processes occur behind the scenes and are not directly observable. As a result, traditional lecture-based instruction may not provide sufficient opportunities for students to develop the critical thinking and troubleshooting skills required in networking careers. To be successful, students need to develop both a strong theoretical understanding of networking concepts and the practical competencies necessary to configure, manage, and troubleshoot network systems.

The Cisco Networking Academy seeks to address these educational challenges by employing a blended learning approach that integrates theoretical instruction with simulations, assessments, and practical laboratory exercises (Cisco, 2024). This combination of instructional methods is intended to strengthen students' conceptual understanding while providing meaningful opportunities for hands-on practice. However, despite the widespread adoption of the Cisco Networking Academy in higher education, limited qualitative research has examined how professors at U.S. colleges and universities perceive the platform's effectiveness in supporting computer science students enrolled in introductory networking courses.

4 Purpose of the Study

The purpose of this study is to examine the effectiveness of the Cisco Networking Academy in teaching introductory networking courses to computer science students in the United States. Specifically, the study explores faculty members' perceptions of the platform's effectiveness in supporting student learning and enhancing the overall instructional experience. Understanding instructors' perspectives provides valuable insights into the strengths and limitations of integrating Cisco Networking Academy into undergraduate networking education.

The study also investigates the educational benefits of using Cisco Packet Tracer and virtual laboratory environments to support hands-on learning and reinforce networking concepts. In addition, it examines the challenges instructors encounter while implementing the platform, including instructional, technical, and pedagogical issues that may affect its successful integration into the curriculum.

Furthermore, the research explores how the Cisco Networking Academy influences student engagement and learning outcomes in introductory networking courses. Finally, the study seeks to identify recommendations from faculty members for improving networking instruction and enhancing the effectiveness of the Cisco Networking Academy as a teaching and learning resource in computer science programs.

5 Research Questions

This study addresses the following research questions:

1. How do professors perceive Cisco Networking Academy as a teaching platform for introductory networking courses?

2. What instructional benefits does Cisco Academy provide for computer science students?

3. What challenges do instructors encounter when implementing Cisco Academy curricula?

4. How does Packet Tracer contribute to student learning and engagement?

5. What improvements do instructors recommend for networking education using Cisco Academy?

6 Methodology

6.1 Research Design

This study used a qualitative research methodology to examine instructors' experiences with Cisco Networking Academy. Qualitative methods were selected because they provide detailed insights into instructional practices, faculty perceptions, and educational experiences [4]. The study employed Semi-structured interviews, Open-ended questionnaires, and Thematic analysis.

6.2 Participants

Participants consisted of professors and instructors from colleges and universities in the United States who teach introductory networking courses using Cisco Networking Academy materials.

6.3 Selection Criteria

Participants were selected using purposeful sampling based on specific inclusion criteria relevant to the study's objectives. To be eligible, participants had to be currently teaching networking courses using the Cisco Networking Academy curriculum. In addition, they were required to teach computer science or information technology students and have a minimum of one year of teaching experience using the Cisco Networking Academy platform. These criteria ensured that participants possessed sufficient experience to provide meaningful insights into the effectiveness of the curriculum and instructional tools. A total of 12 instructors who met these eligibility requirements participated in the study. Their experience teaching networking courses through the Cisco Networking Academy provided valuable perspectives on the platform's strengths, challenges, and impact on student learning in higher education.

6.4 Data Collection

Data was collected through virtual interviews and email questionnaires to gather detailed insights into instructors' experiences using the Cisco Networking Academy in introductory networking courses. The data collection instruments were designed to explore multiple aspects of teaching and learning, including instructional strategies, student engagement, laboratory activities, the use of Cisco Packet Tracer, and the challenges instructors encountered while implementing the curriculum. Participants were also invited to provide recommendations for improving the effectiveness of networking instruction.

The interview protocol included several open-ended questions that encouraged participants to share their experiences and perspectives. Sample questions included: *How effective is the Cisco Networking Academy for introductory networking instruction? What benefits do students gain from using Cisco Packet Tracer? What challenges do students encounter while using the platform? How do you supplement the Cisco Networking Academy materials in your courses?* and *What improvements would you recommend to enhance networking instruction?* These questions provided rich qualitative data that supported an in-depth analysis of faculty perceptions regarding the use of the Cisco Networking Academy in higher education.

6.5 Data Analysis

Interview responses were analyzed using thematic analysis. The researcher identified recurring themes across instructor responses, categorized findings, and interpreted patterns related to networking education effectiveness [1].

7 Findings

7.1 Theme 1: Hands-On Learning Improves Student Engagement

Nearly all instructors emphasized the importance of hands-on activities in networking education. Participants stated that students learn networking concepts more effectively when they actively configure routers, switches, and network services using Packet Tracer. One instructor explained "Students understand routing protocols much faster when they configure them themselves." Another instructor stated "Packet Tracer allows students to experiment without fear of breaking real equipment." These findings align with previous research emphasizing experiential learning in technical education [5].

7.2 Theme 2: Packet Tracer Enhances Visualization

Instructors consistently identified Cisco Packet Tracer as one of the most valuable components of the Cisco Networking Academy. Faculty members explained that students frequently struggle to understand abstract networking concepts, including routing tables, packet forwarding, virtual local area network (VLAN) segmentation, subnetting, and network protocols. These topics are often difficult to comprehend through traditional lectures alone because many networking processes occur behind the scenes and cannot be directly observed.

According to the participants, Cisco Packet Tracer helps bridge this gap by allowing students to visualize packet movement, network topologies, device communication, and protocol behavior in a simulated environment. The interactive simulations enable students to observe how networking devices interact and how configuration changes affect network performance, thereby reinforcing theoretical concepts through practical application.

Several instructors also reported that the simulation-based environment significantly reduces student anxiety during laboratory activities. Because students can experiment, make mistakes, and troubleshoot network configurations without the risk of damaging physical equipment, they become more confident in applying networking concepts. This increased confidence encourages active learning, improves problem-solving skills, and better prepares students for hands-on work with real networking hardware.

7.3 Theme 3: Structured Curriculum Supports Instruction

Professors expressed strong appreciation for the organization and structure of the Cisco Networking Academy materials. Participants indicated that the platform provides a well-developed instructional framework that helps instructors effectively deliver introductory networking courses. Faculty members highlighted several advantages of using Cisco Academy, including reduced course preparation time, standardized assessments, interactive learning modules, and support for asynchronous learning environments.

One participant described the curriculum by stating, "The curriculum provides a complete instructional framework." Another instructor emphasized the value of the platform for faculty members who are new to teaching networking courses, explaining that "Cisco Academy gives new instructors a strong starting point." These perspectives suggest that the structured content and resources provided by Cisco Academy can assist instructors in developing and managing networking courses more efficiently.

In addition, participants viewed Cisco Academy's online learning management environment as particularly beneficial for hybrid and fully online teaching

formats. The availability of digital learning resources, assessments, and interactive activities allows instructors to provide flexible learning opportunities while maintaining consistency in course delivery and student evaluation.

7.4 Theme 4: Students Require Supplemental Materials

Although instructors valued the resources provided by Cisco Networking Academy, many participants indicated that the curriculum alone is not sufficient to fully support student learning in introductory networking courses. Faculty members reported supplementing Cisco Academy materials with additional instructional resources, including expanded lectures, real-world case studies, open-source networking tools, Linux-based networking laboratories, and cybersecurity-related exercises. These supplemental activities were viewed as important for helping students develop a broader understanding of networking principles and their practical applications.

Several instructors expressed concerns that the curriculum may place too much emphasis on Cisco-specific technologies and configurations rather than focusing more broadly on fundamental networking concepts. While Cisco-focused skills are valuable for students pursuing industry certifications and networking careers, participants emphasized the importance of ensuring that students understand the underlying principles behind networking operations.

One participant highlighted this concern by stating, "Students still need conceptual explanations beyond command memorization." This perspective reflects the belief that effective networking education should balance vendor-specific technical skills with deeper conceptual knowledge, critical thinking, and problem-solving abilities that can be applied across different networking environments and technologies.

7.5 Theme 5: Certification Alignment Motivates Students

Many instructors reported that students value the opportunity to develop skills that align with professional certifications. Faculty members indicated that certification-oriented instruction provides several benefits, including improving students' career readiness, increasing motivation, enhancing employability, and building confidence in their technical abilities. By connecting academic coursework with industry-recognized credentials, students are able to see the practical relevance of networking concepts and their potential applications in professional settings.

Cisco (2024) reported that many students who complete certification-aligned courses achieve employment opportunities or continue their education in technology-related fields. Several instructors noted that students perceive Cisco Certified Network Associate (CCNA) preparation as particularly valuable because it

provides a pathway toward internships and entry-level information technology positions.

Participants emphasized that the integration of certification preparation into networking courses helps bridge the gap between academic learning and industry expectations. By gaining exposure to certification objectives and hands-on networking skills, students develop greater confidence in their abilities and become better prepared to pursue careers in networking and related IT fields.

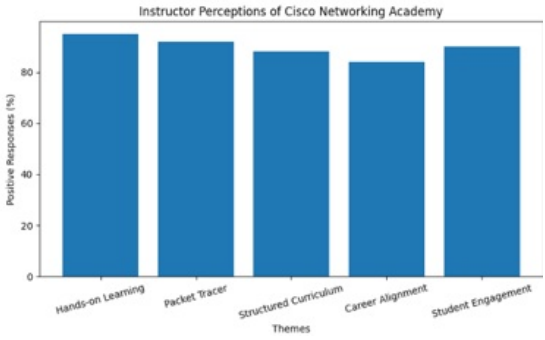


Figure 3: Instructor presentations of Cisco Academy

8 Discussion

The findings indicate that Cisco Networking Academy significantly supports networking education within computer science programs. Participants identified several strengths of the platform, including hands-on learning opportunities, simulation-based instruction, a structured curriculum, alignment with industry requirements, and accessibility for both instructors and students. Among these features, Cisco Packet Tracer emerged as the most influential educational tool because it effectively bridges the gap between theoretical networking concepts and practical implementation.

The study further confirms that networking education benefits from active learning approaches. Students develop a stronger understanding of networking concepts when they actively configure, analyze, and troubleshoot network environments rather than relying solely on traditional lecture-based instruction. By engaging in practical activities, students are able to apply theoretical knowledge, develop problem-solving skills, and gain confidence in their technical abilities. However, the findings also reveal several limitations of using Cisco Networking Academy as a standalone instructional resource. Many instructors reported the need to supplement Cisco Academy materials because some net-

working concepts require deeper theoretical explanations, students need exposure to networking environments beyond Cisco-specific technologies, and real-world networking practices extend beyond certification objectives. These findings suggest that Cisco Networking Academy should be used as a foundational instructional platform that is enhanced with additional resources, activities, and teaching strategies to provide a more comprehensive networking education experience.

8.1 Implications for Computer Science Education

The results of this study provide several important implications for computer science education. From a curriculum development perspective, universities should consider integrating simulation-based laboratories, practical troubleshooting exercises, industry-aligned certifications, and active learning methodologies into introductory networking courses. These approaches can help students develop both conceptual understanding and practical technical skills that align with academic and industry expectations.

8.2 Faculty Development

Institutions should provide faculty development opportunities to help instructors effectively utilize Cisco Networking Academy resources and improve networking instruction. Training should focus on the effective use of Packet Tracer, laboratory-based teaching strategies, active learning approaches, and methods for aligning course content with industry certification objectives. Well-prepared instructors are better positioned to maximize the educational benefits of the platform while addressing its limitations.

8.3 Student Engagement and Learning Outcomes

The study demonstrates that hands-on networking activities contribute to improved student engagement and learning outcomes. Practical exercises help increase student confidence, strengthen technical competency, enhance problem-solving skills, and improve motivation. By allowing students to actively participate in network configuration and troubleshooting activities, instructors can create more meaningful and effective learning experiences.

8.4 Industry Preparation

Cisco Networking Academy plays an important role in helping bridge the gap between academic preparation and workforce expectations. By exposing students to practical networking technologies, industry standards, and

certification-aligned skills, the platform provides students with valuable experiences that support career readiness. When combined with broader conceptual instruction and additional hands-on activities, Cisco Academy can serve as an effective component of undergraduate networking education.

9 Limitations of the Study

Several limitations of this study should be acknowledged when interpreting the findings. First, the sample size was relatively small, which may limit the generalizability of the results to a broader population of networking instructors. Second, participants were limited to instructors from institutions within the United States, and their perspectives may not fully represent experiences in other educational contexts or geographic regions. Third, the study focused primarily on faculty perspectives and did not directly include student experiences, perceptions, or feedback regarding the use of Cisco Networking Academy. Finally, quantitative measures of student performance, such as assessment scores, course outcomes, or certification success rates, were not analyzed as part of this research.

Future research should expand upon this study by incorporating larger participant groups to provide a broader representation of instructor experiences. A mixed-methods research approach that combines qualitative faculty perspectives with quantitative student performance data could provide a more comprehensive understanding of Cisco Networking Academy's effectiveness. Additional studies should also examine student learning outcomes, engagement levels, and skill development through direct performance analysis.

Furthermore, comparative research examining Cisco Networking Academy courses alongside non-Cisco networking curricula would provide valuable insights into the relative effectiveness of different instructional approaches. Such studies could help identify which components of networking education, including simulations, laboratory activities, certification alignment, and active learning strategies, contribute most significantly to student success.

10 Recommendations

Based on the findings of this study, several recommendations are proposed to enhance the effectiveness of introductory networking education using Cisco Networking Academy. First, instructors should integrate Cisco Academy materials with broader conceptual networking instruction. While the platform provides extensive hands-on learning opportunities and certification-oriented content, students benefit from additional explanations that emphasize fundamental networking principles and the theoretical concepts underlying network

operations.

Second, Cisco Packet Tracer should be used extensively throughout introductory networking courses to reinforce classroom instruction. Its simulation-based environment enables students to practice network design, configuration, and troubleshooting in a safe and cost-effective setting, thereby improving technical competency and confidence. In addition, instructors should incorporate collaborative laboratory activities that encourage teamwork, communication, and problem-solving skills, reflecting the collaborative nature of real-world networking environments.

Furthermore, networking curricula should include exposure to non-Cisco networking technologies and vendor-neutral concepts to provide students with a broader understanding of modern networking environments. Although Cisco technologies are widely used in industry, familiarity with multiple networking platforms and standards better prepares students for diverse professional roles. Instructors should also provide additional academic support for students who struggle with networking concepts through supplemental tutorials, guided laboratory exercises, office hours, and other learning resources designed to improve student success.

Finally, networking courses should incorporate project-based assignments that require students to apply their knowledge to realistic networking scenarios. Such projects promote critical thinking, troubleshooting, and practical application of networking concepts while helping students develop the skills expected by employers. At the same time, instructors should strive to balance certification preparation with a strong emphasis on theoretical understanding, ensuring that students not only learn how to configure networking devices but also understand the underlying principles that govern network design, communication, and operation. This balanced approach will better prepare graduates for both professional certification and long-term success in networking and information technology careers.

11 Conclusion

Cisco Networking Academy has become one of the most influential platforms in networking education. Its combination of simulations, hands-on laboratories, structured curriculum, and certification alignment makes it highly effective for teaching introductory networking courses to computer science students.

Qualitative findings from instructors in the United States indicate that Cisco Academy improves student engagement, practical skills, and career readiness. Packet Tracer, in particular, plays a critical role in helping students visualize and understand networking concepts.

Despite some limitations, Cisco Networking Academy provides substantial

educational value when combined with instructor guidance, supplemental materials, and active learning strategies.

As networking technologies continue evolving, educational institutions must adopt innovative teaching methods that prepare students for modern computing environments. Cisco Networking Academy represents a strong foundation for experiential networking education in computer science programs.

References

[1] Virginia Braun and Victoria Clarke. "Using thematic analysis in psychology". In: *Qualitative research in psychology* 3.2 (2006), pp. 77–101.

[2] Cisco. *Cisco Networking Academy*. 2024. URL: https://www.netacad.com (visited on 07/16/2026).

[3] Cisco Systems, Inc. *Learning and Digital Skills: Cisco Networking Academy Impact*. 2025. URL: https://www.cisco.com/c/m/en_us/about/purpose/reporting-hub/community-resilience/digital-skills.html (visited on 02/16/2026).

[4] John W Creswell and J David Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 5th ed. Thousand Oaks, CA: SAGE Publications, 2018. ISBN: 9781506386706.

[5] Kristen E DiCerbo. "Hands-on instruction in the cisco networking academy". In: *2009 Fifth International Conference on Networking and Services*. IEEE. 2009, pp. 581–586.

[6] Mohd Syahrizad Elias and Ahmad Zamzuri Mohamad Ali. "Survey on the challenges faced by the lecturers in using packet tracer simulation in computer networking course". In: *Procedia-Social and Behavioral Sciences* 131 (2014), pp. 11–15.

[7] Meredith Liu. "Cisco Networking Academy: Next-Generation Assessments and Their Implications for K-12 Education." In: *Clayton Christensen Institute for Disruptive Innovation* (2014).

[8] Arjun Singh Saud, Binod Adhikari, and Subeksha Piya. "Evaluating the Effectiveness of Cisco Packet Tracer for Teaching Networking Concepts at the Undergraduate Level". In: *Perspectives on Higher Education* 15.01 (2025), pp. 123–136.

Alternative Grading in Introductory Computer Science Courses: Practices, Outcomes, and Challenges*

William Turner
Mathematics and Computer Science
Wabash College
Crawfordsville, IN 47933
turnerw@wabash.edu

Abstract

This paper presents a reflective case study of my use of alternative grading in two introductory computer science courses at Wabash College. I describe the pedagogical tensions that led me to adopt alternative grading, outline the grading systems I have used, and reflect on their effects on student performance and course design. My experience suggests that alternative grading can better align course grades with demonstrated understanding, while also introducing challenges related to student pacing, attendance, and course logistics.

1 Introduction

Throughout my 32 years of teaching mathematics and computer science courses at the collegiate level, I have felt a persistent tension between giving students opportunities to learn from their mistakes and ensuring that they genuinely understand the material. While teaching mathematics courses in graduate school, I regularly allowed students to rework exams to recover some of the points they had missed. I did this to encourage them to revisit difficult material, to learn

from their mistakes, and to prepare for the final exam. However, some students were able to barely pass the course through recovered exam points, homework, and other out-of-class work, even though their in-class scores suggested they never fully grasped the material.

I tried several ways to adjust the grading system, including changing the relative weights of exams, homework, and reworked exams in the final grade. I also considered using a geometric mean to average the scores. However, none of these approaches solved the underlying problem. The challenge was to allow students to recover from early mistakes while ensuring high course grades still reflected real understanding of the material. The alternative grading systems described in this paper emerged from that challenge.

1.1 Introductory Computer Science at Wabash College

I was hired at Wabash College to teach both mathematics and computer science and eventually run the computer science program. One of the original courses in the curriculum was CSC 111: Introduction to Programming. It is a fairly standard first-semester programming course, but it also includes some theoretical components such as the P vs. NP problem. However, during my first few years on the faculty, we noticed a drastic decline in enrollment in computer science courses to the point that fewer than ten students had enrolled in CSC 111 for four consecutive semesters. We believed that several factors contributed to this decline, including the removal of a computer science requirement from other majors and the increased computer literacy of incoming students, who may not have felt a need to take a programming course. We also saw some students struggling with basic programming concepts such as control structures. To address these issues, we introduced a new course, CSC 101: Introduction to Computer Science, to give students another opportunity to fulfill their quantitative studies distribution requirement. The course was based on a "CS0" model that I first encountered in a course offered at Duke University. We hoped it would attract students who would not otherwise take a computer science course. We also believed it would strengthen the rest of the computer science curriculum by removing the non-programming aspects from CSC 111 and allowing that course to focus more fully on the increasingly complex features of modern programming languages.

Over the years, I have taught both courses several times. For many years, I was the primary instructor for CSC 101, while a junior colleague with a background in numerical analysis taught CSC 111. About a decade ago, another colleague with an interest in machine organization took over more of the CSC 101 duties, and I began teaching CSC 111 more regularly. However, the tensions I felt in mathematics courses in graduate school persisted, and the emphasis in CSC 111 on programming projects over written exams made some

of the tensions more acute.

1.2 Alternative Grading

I was first introduced to alternative grading through Linda Nilson's book *Specifications Grading* [6] and Robert Talbert's blog [7]. Specifications grading offered a solution to these long-standing tensions that differed significantly from the approaches I had previously considered. Instead of worrying about the relative weights of assignments, I could focus on whether students satisfied each assignment individually. Students could reattempt assignments, and final grades reflected what they completed by the end of the semester. I began to see what would later be called the Four Pillars of Alternative Grading [3, 8]. I found this method worked well for assignments such as programming projects.

Another alternative grading method, standards-based grading [1], focuses on students demonstrating their understanding of individual skills or topics. In a variation called standards-based testing, students demonstrate their understanding through exams or quizzes. As with other alternative grading methods, there are many variations such as scoring each exam on how many standards a student passes [2] or basing the final course grade on the number of standards passed.

The alternative grading community is growing, and many resources for alternative grading methods exist, including books [3, 6], blogs [4], conferences [5], and podcasts [9].

2 Approach

My original approach to alternative grading in CSC 111 in Fall 2018 was heavily influenced by specifications grading (§2.1). However, I found the system cumbersome and confusing to students, so I later modified it into a more streamlined approach (§2.2).

I had also tried adapting this original approach for courses that traditionally relied less on programming projects and more on exams, such as CSC 101, theoretical computer science courses, and mathematics courses, but I found the result awkward. The inaugural Grading Conference [5] in 2020 introduced me to standards-based testing, which I implemented in CSC 101 (§2.3) as well as other courses.

For the past few years, I have used a blend of specifications grading and standards-based grading in my courses. I added standards-based testing components to CSC 111, and I brought specifications grading into mathematics and theoretical computer science courses through writing assignments that functioned similarly to the weekly programming projects in CSC 111. I also added other components to encourage students to prepare for class, to take advantage

Table 1: CSC 111 Fall 2018 Grading Scheme

	A	B	C	D	F
Class meetings (28 total)	26	24	22	20	-
Classroom activities (28)	26	24	22	20	-
Chapter reading quizzes	90%	80%	70%	60%	-
Projects: S or E (14 total)	12	11	10	9	-
Projects: E	9	7	5	-	-

of resources such as office hours and tutoring services, and to participate in the wider activities of the department (§2.4).

2.1 Original Approach

Grades in CSC 111 in Fall 2018 were based on four criteria: class attendance, classroom activities, reading quizzes, and weekly programming projects (Table 1). To earn a given course grade, a student had to satisfy all specifications in that grade's column. Quizzes were scored with a traditional point system, classroom activities were scored pass/fail, and weekly programming projects were scored on a four-level scale based on both the project itself and the written report required with each project:

E: The project satisfies the specifications, and the report clearly describes the overall program structure, how to run the program, how the program was created and tested, and any known problems. It also reflects what was learned through the project.

S: The project satisfies the specifications, and the report clearly describes how to run the program, but it may have omissions for the other requirements or have significant writing errors.

P: The project does not satisfy the specifications, but it demonstrates a partial understanding of concepts, or the report has more significant omissions.

I: The submission contains significant omissions. There is not enough information to determine whether the concepts are fully understood.

The last two rows of Table 1 show that to earn a grade of "C" or higher, students needed some projects to earn an "E" level and not just an "S" level. For example, out of fourteen total projects during the semester, an "A" grade required a student to submit at least twelve "S"- or "E"-level projects, of which at least nine needed to earn an "E".

Table 2: CSC 111 Fall 2020 Grading Scheme

	A	B	C	D	F
Projects (14 total)	13	12	10	9	-

In addition, students began the semester with four virtual tokens, called "Oops Bits", that students could use to resubmit an incomplete (but submitted) assignment, make up a missed quiz or in-class assignment, gain a 24-hour extension on a due date of an assignment, or miss an extra day of class. Students found these tokens confusing, particularly in conjunction with the four-level project scoring scale. Because I limited how long after the original due date students could resubmit a project, some students held onto the tokens and did not use them when they would have helped their grades. In particular, students who earned an "S" on a project sometimes hesitated to use a token to resubmit and earn an "E" in case they needed the token later to convert a "P" or "I" into an "S" or "E".

2.2 Modified Specifications Grading

Because students found the original system difficult to navigate, I simplified the grading structure when I next taught CSC 111 by eliminating the attendance, quiz, and classroom assignments and basing the final grade only on the number of weekly projects successfully completed (Table 2). I also eliminated the "Oops Bits" tokens and allowed students to submit any two projects each week, which gave the students more flexibility without the hesitation introduced by the token system. On the other hand, I reduced the project scale to two levels, with successful completion now corresponding to the former "E" level. This raised the standard for what counted as acceptable project work. Students could now miss only one project and still earn an A, rather than two under my original approach.

2.3 Standards-Based Testing

In courses that relied more on exams than on projects, I implemented a system of standards-based grading. Each class meeting focused on a learning target, and students demonstrated mastery of these targets on exams, called "mastery opportunities", administered every other week throughout the semester. Each problem corresponds to a learning target, and each exam is cumulative in that it gives students an opportunity to demonstrate mastery of any targets they have not yet passed. As a result, the first exam is roughly the same length as a typical exam, but later exams contain problems from the entire course up

Table 3: CSC 101 Fall 2023 Grading Scheme

	A	B	C	D	F
Learning Targets (out of 33 total)	30	27	24	20	-

Table 4: CSC 101 Spring 2026 Grading Scheme

	A	A-	B+	B	B-	C+	C	C-	D	F
Learning Targets (32)	30	29	28	27	26	25	23	22	19	-
Online Homework							75%		60%	-
Computational Life	4			3			2		1	-
Resource Utilization									1	-

to that point. In this way, the last exam allows students to demonstrate their knowledge of all learning targets from the beginning of the semester. The final exam is one last three-hour opportunity to demonstrate mastery of all learning targets. Students' final grades were determined exclusively by the number of learning targets they passed by the end of the semester (Table 3).

2.4 Current Approach

For the past few years, I have used a mixed approach that combines specifications grading for projects in CSC 111 with standards-based testing for both CSC 101 (Table 4) and CSC 111 (Table 5). In addition, students must complete work in three other categories. Online Homework gives instant feedback and helps prepare students for exams. Resource Utilization encourages students to take advantage of office hours and the tutoring center. Computational Life encourages students to attend events such as the department colloquium series and computer science club activities. It also helps the course satisfy Wabash's credit-hour policy, which requires additional faculty-directed instruction beyond the regular class meetings so that the course meets the federal standard for a four-credit-hour course despite only meeting for 150 minutes per week.

Table 5: CSC 111 Spring 2026 Grading Scheme

	A	A-	B+	B	B-	C+	C	C-	D	F
Learning Targets (20)	19	18	17	16	15	14	13	12	11	-
Projects (13)	12		11		10		9	8	7	-
Online Homework							75%		60%	-
Computational Life	4			3			2		1	-
Resource Utilization									1	-

3 Reflections

After eight years of using alternative grading methods—and after teaching nearly one-third of my students at Wabash College in such courses—I have noticed several patterns.

3.1 Student Grades

The first observation is difficult to quantify: I had greater confidence that students who passed the course actually understood the material because I could point to specific problems and assignments on which they demonstrated that understanding. This increased confidence did not come at the cost of lower grades overall.

Quantitatively, alternative grading did not reduce student performance overall, but it did change the grade distribution in notable ways. The average course GPA in courses using alternative grading is slightly higher than in courses using traditional grading: 2.674 vs. 2.567. As Figure 1 shows on page , the most striking difference in the grade distributions for all of the courses I have taught while at Wabash College is the much larger share of A grades in alternative grading sections, while several middle grade categories appear less frequently. Figure 2 on page shows an even more pronounced change in CSC 101 and CSC 111, where sections with alternative grading methods have 2.27 times as many A grades as traditional grading sections, compared to 2.00 times as many for all courses. One possible contributing factor is that the system does not explicitly penalize students for learning the material later in the course. Another possible factor is that students can see more clearly what they must accomplish to earn a particular grade, which may give them a boost of confidence late in the semester.

Alternative grading is also associated with increased numbers of F grades in all courses and D grades in CSC 101 and CSC 111. The increase in D grades comes primarily from CSC 101, as shown in Figure 3 on page . This may reflect the fact that many students take the course for distribution (or general education) credit and do not intend to continue on to other computer science courses. I suspect this increase results from a combination of students aiming only for a passing grade and the relative ease with which they can calculate the minimum work needed to achieve it. Another factor may be the CC/NC mechanism the college implemented a decade ago that allows underclassmen to earn credit in courses without affecting their GPA, but at the cost of not allowing the credit to count for a major, minor, or prerequisite. This could particularly affect courses that students take primarily for distribution credit. CSC 111 shows a decrease in D grades and an increase in F grades (Figure 4). However, students who fail typically do not even submit enough work to pass

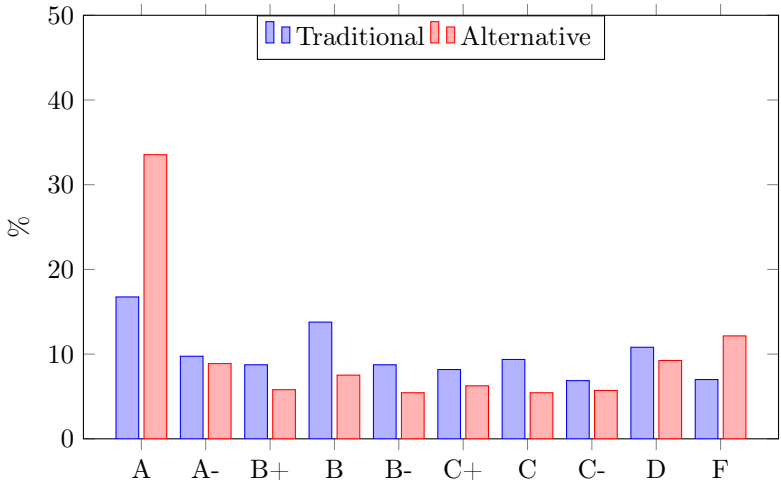


Figure 1: All Courses Grade Distribution

the course, or at best they submit just enough work but not at a satisfactory level. Most do not attend the final exam. In general, students who perform poorly do not take advantage of the opportunities to revise and resubmit work after receiving feedback.

3.2 AI

With the recent rise of AI tools, faculty are concerned about students using AI to complete assignments. Faculty in many disciplines have discussed moving away from take-home assignments and even re-implementing bluebook exams that had all but disappeared from Wabash. While I know some students are using AI on programming projects—and I have submitted several academic dishonesty strikes for such cases—I am less concerned that students will perform well in the course overall without understanding the material. To pass the course, they must still demonstrate knowledge of the learning targets on in-person exams. Performing well on projects cannot balance out low exam scores.

3.3 Disadvantages

While I see many advantages to alternative grading methods, perhaps the biggest disadvantage is that some students treat alternative grading as a license

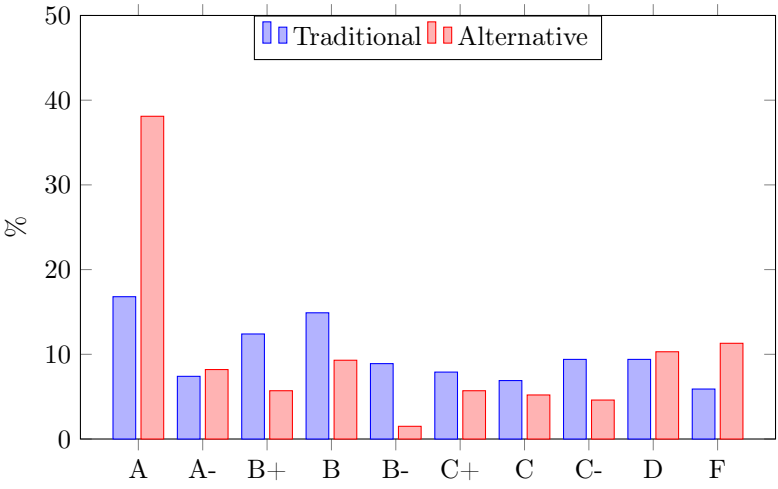


Figure 2: CSC 101 and 111 Grade Distribution

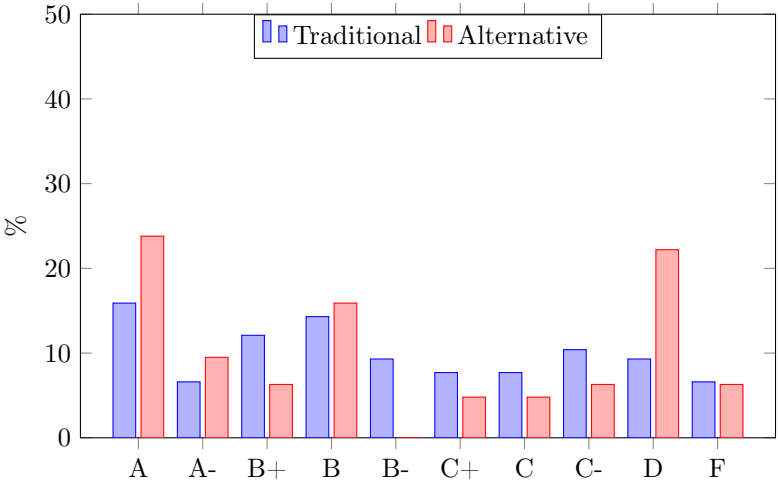


Figure 3: CSC 101 Grade Distribution

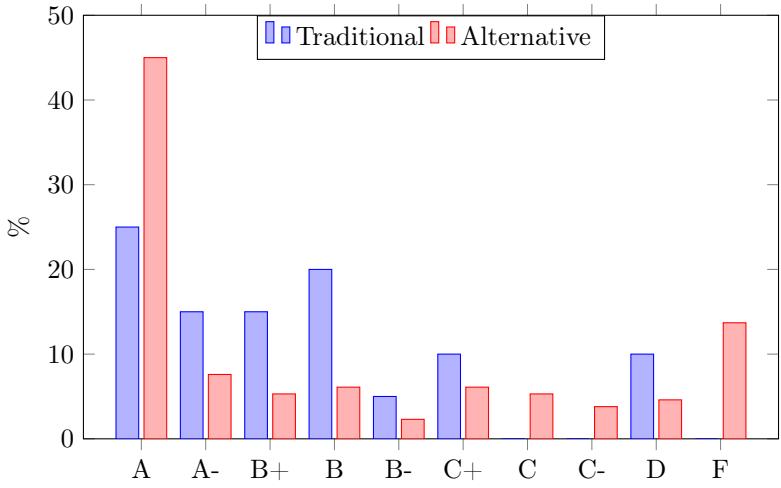


Figure 4: CSC 111 Grade Distribution

to procrastinate. Instead of submitting assignments early so that they can receive feedback and learn from their mistakes, some students believe that they can wait until late in the semester to engage seriously with the material. My original system, which required students to submit projects by the original due date and limited resubmissions, may have reduced this problem. However, I would rather not restrict resubmissions so sharply nor require students to submit work before they are ready. I try to be proactive by monitoring which students are not submitting projects and encouraging them to work on them. I use the college's early warning system to notify academic advisors and coaches when students have not submitted any projects for several weeks. However, some students fail to heed the warnings.

I have also seen declining attendance. While alternative grading methods could contribute to the problem, anecdotal evidence suggests the problem is widespread throughout the college, in traditionally graded courses as well. The problem appears to have worsened after COVID and to have become more acute recently.

4 Possible Modifications

One practice I could adopt in the future would be to require all students to take the final exam. At present, students take it only to earn learning targets

they have not yet passed, but it could retest them on targets they have already passed. Earning a particular grade could require students both to pass a certain number of learning targets on the final exam and to reach a total number of passed learning targets throughout the course.

To combat absenteeism, I could reinstate an attendance policy or in-class assignments from my original implementation. Either of these would require additional work to accommodate excused absences. This would particularly affect athletes, who make up a high percentage of Wabash's student body. On the other hand, in-class assignments could encourage students to prepare before class.

My experience suggests that alternative grading can provide a more accurate picture of student learning in introductory computer science courses, particularly when course grades depend on demonstrated mastery rather than accumulated points. At the same time, these systems require careful design to address student procrastination, attendance, and logistical complexity. Continued refinement of these approaches may help preserve their benefits while reducing their drawbacks.

References

[1] David Clark. *Common questions about standards-based grading*. 2024. URL: https://gradingforgrowth.com/p/common-questions-about-standards.

[2] David Clark. *Standards-Based Testing*. 2023. URL: https://gradingforgrowth.com/p/standards-based-testing.

[3] David Clark and Robert Talbert. *Grading for Growth: A Guide to Alternative Grading Practices that Promote Authentic Learning and Student Engagement in Higher Education*. Routledge, 2023. ISBN: 978-1642673814.

[4] *Grading for Growth*. URL: https://gradingforgrowth.com/.

[5] The Center for Grading Reform. *The Grading Conference*. URL: https://www.centerforgradingreform.org/grading-conference/.

[6] Linda B. Nilson. *Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time*. Routledge, 2014. ISBN: 978-1620362419.

[7] Robert Talbert. *Specifications grading: We may have a winner*. 2018. URL: https://www.rtalbert.org/blog-archive/index.php/2017/04/28/specs-grading-iteration-winner.

[8] Robert Talbert. *The Four Pillars of Alternative Grading*. 2025. URL: https://gradingforgrowth.com/p/the-four-pillars-of-alternative-grading.

[9] *The Grading Podcast*. URL: https://thegradingpod.com/.

Electronic Textiles: Physical Computing in the Time of Artificial Intelligence*

Iris Howley
Interdisciplinary Computing Program
Kenyon College
Gambier, OH 43022
howley1@kenyon.edu

Abstract

While much effort is currently dedicated to responding to increasing student use of generative artificial intelligence (AI) in the college classroom, I propose a curriculum of physical computing to emphasize the aspects of computing that AI cannot yet impact. In this article, I outline an Electronic Textiles course that uses the LilyPad Arduino ecosystem to encourage engagement with computer science concepts from students who do not [yet] view themselves as computer scientists. While the curriculum is shown to be effective in teaching foundational programming concepts as well as engaging new student populations with computing, one of the greatest challenges of this course is adapting it for students of varying technical backgrounds and uptake.

1 Introduction

During this current Artificial Intelligence (AI) hype cycle, instructors and learners alike are questioning computer programming's place in the computer science curriculum [4]. Thoughtfully integrating generative AI tools into introductory programming courses has shown great potential, whether through emulating 1-on-1 tutors [5] or as coding assistants for fixing errors and explaining algorithmic concepts [7]. However, these tools and pedagogical ideology leave

Table 1: Course topics by week

Week	Topics
1	Internet and Websites (Jekyl Template for Project Portfolio)
2	Sewing, Hello World, Modular arithmetic
3	Conditionals, Circuits
4	Nested Conditionals, Conductivity
5	While loops, Arduino
6	For loops, State-based programming
7	Random, Digital Inputs, RGB component
8	Light/Temperature Sensor, Buzzer (Speakers)
9	Void functions, Value-returning functions
10	Arrays, Project Proposals
11	Continuing in Computer Science, Variable scope
12	Final Project Demonstrations

untouched an entire world of computing that AI tools cannot yet fully influence, that of physical computing platforms [6], such as wearables. In this article, I explore a sample computing curriculum for an introductory computer science course that introduces students to programming through the design and implementation of wearable Arduino components.

The main goal of the Electronic Textiles course described in this article is to engage students who would not ordinarily view themselves as computer scientists. while providing a solid background in computational thinking, many major programming concepts, and the design process, all under the guise of eventually creating our own personal projects. To achieve this goal, I selected the LilyPad Arduino ecosystem for its ease of use, no need to calculate resistance for the circuits, plentiful online support, as well as already established success in engaging different populations with computer programming [3].

2 Electronic Textiles Course Curriculum

The basic structure of the Electronic Textiles course for the past four offerings, as shown in Table 1, generally starts with a week of discussing the Internet and setting up blogs to serve as project portfolios, followed by 4 weeks of C-programming (as the Arduino programming language is a subset of C++), interspersed with sewing and circuitry concepts. The remainder of the course introduces various new Arduino components that must be combined with prior knowledge to design desired behaviors. These all lead up to student-designed projects presented on the last day of class.

1. Closely examine the C program below.

C Program

```
#include <stdio.h>
int main()
{
    int flashlt = 0; // Is flashlight on or not?
    int lighting = 0; // What level of lighting is there?

    if (flashlt == 0 && lighting < 0) {
        printf("Too dark. Turn on the MEGA flashlight.");
    } else if (flashlt == 0 && lighting < 50) {
        printf("Turn on flashlight.");
    } else if (flashlt == 0) {
        printf("Do you really need a flashlight?");
    } else if (flashlt == 1 && lighting > 50) {
        printf("SO MUCH BRIGHTNESS.");
    }
    return 0;
}
```

a. Come up with values for `flashlt` and `lighting` that will test *each* part of the conditional. List five combinations of numbers to test and predict what will be printed.

flashlt	lighting	Predicted Output

b. Circle the code phrases (i.e., multiple keywords) that are *repeated* in the code above.

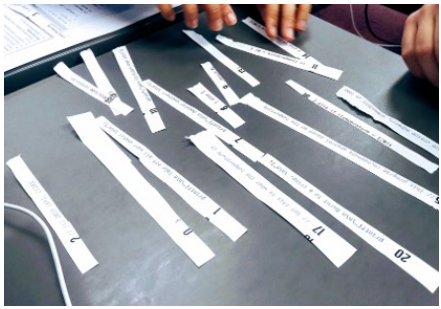


Figure 1: An example concept model from a POGIL on nested conditionals on the left, and mid-process Parsons problem solution to a homework on the right.

The class consists of 18 students at a private liberal arts college in New England, with a mixture of students across all four class years. It meets twice per week for 75-minutes and an assignment is due nearly every class period. A typical class period incorporates various pedagogical approaches, including an adapted Process-Oriented Guided Inquiry Learning (POGIL) [2] method, Parsons problems [1], lecture, student presentations, and open lab time. Examples of the POGIL and Parsons problems approaches are shown in Figure 1. As there does not yet exist a textbook to support this curriculum, course slides are shared, and there are also review handouts which provide overviews of programming concepts learned both in C and Arduino, as well as guides for connecting and using the Arduino components.

Assignments are specifically designed to take approximately 2-hours on average, and earlier C-based assignments have patterns that are more useful later on. For example, the "Conditions" assignment has students printing out various messages, depending on a given temperature, which becomes quite useful when the temperature sensor is introduced later.

2.1 Course Projects

Beyond the twice weekly assignments providing small practice in the concepts learned in class, there is a more conceptually complex mid-semester project to learn state-based programming, as well as a student-designed final project. The complexity of the mid-semester project comes from implementing the firefly behaviors illustrated in Figure 2 within the Arduino structure; that is, emulating parallel behaviors in a sequential programming environment (with a `loop()` function that is repeatedly called within a hidden `main()` function).

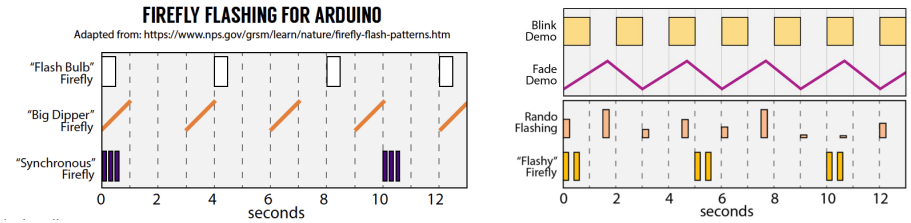


Figure 2: The simultaneous firefly behaviors students programmed for their mid-semester project on the left, and on the right are example state-based blinking patterns implemented together as a class.

This assignment ends up requiring a rather nuanced understanding of modular arithmetic to ensure proper timings of behaviors, but it also requires a shift in perspective of the problem. Rather than using a mindset which focuses on duration, such as "the LED should be on for 500 ms", the programmer needs to instead focus on when a behavior happens, such as "every 500 ms the LED needs to be turned off." Three scaffolding assignments are provided to give students the structure required to think about this problem from a state-based paradigm rather than a sequential paradigm. The state-based paradigm can be useful for students' final projects where they wish to have other parallel-appearing behaviors that are actually happening in a sequential program.

The final project is a student independently-designed project with a set of complexity requirements: each project must have at least two inputs and two outputs, at least one of the inputs must be an analog input (i.e., continuous rather than just off/on) and the same for at least one of the outputs. Beyond these base requirements, students are encouraged to be creative: tell a story, solve a problem, design for others, etc. Example student projects include: a nap mask with timer, light-up musical costumes for pet dogs, musical mug cozy that indicates temperature through music and lights, a purse that alerts to missing objects, a light-up running headband, a goose-shaped meditation timer, a *Stranger Things* "upside down" detector, a knitting aid that uses various blinking patterns to indicate progress, among others.

Prior to presenting their project at demo day, students must first complete a proposal with motivation and materials list, a paper prototype, and photos of alligator clip prototype, as shown in Figure 4. This design process opens up discussions of interaction design processes, as well as cross-disciplinary conversations around paper outlines versus paper prototypes.

Figure 3: Photos of a variety of student projects.

3 Evaluation

In general, students appear to have fun and see great value in the course. Comments such as "Loved this class, very effective at an intersectional approach to my education. Stitching, circuits, coding - everything I am new to and was intimidated by has become familiar now," are fairly typical, with students often recommending the course to their friends. There is typically only one student per semester who decides to continue into our institution's more traditional CS1 course, however, this may also be due to many of the class members being upperclassmen, and the well-known difficulty of the CS1 course. On some occasions, the LilyPad Arduino may show up in projects in other student courses, such as a Theater Major's senior thesis project!

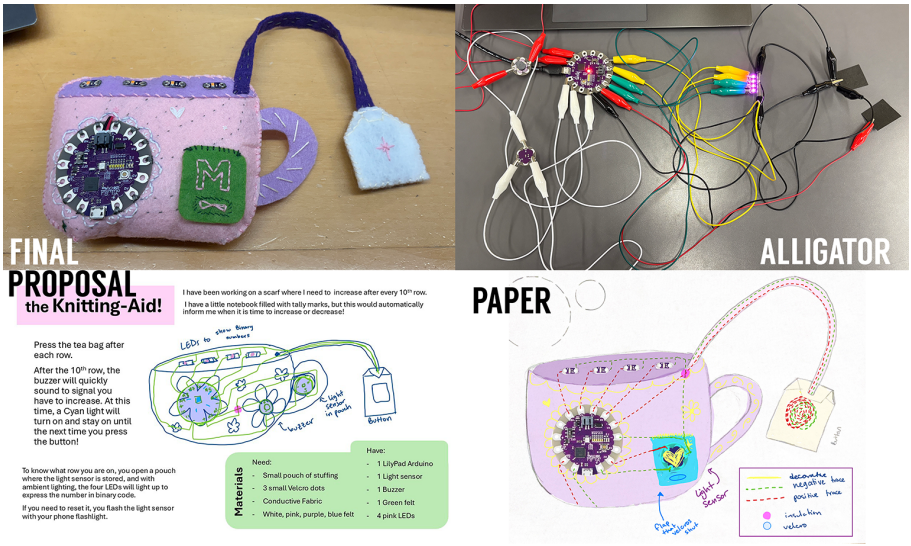


Figure 4: Example of initial pieces of a student final project: proposal, paper prototype, alligator clip prototype, and final product.

Student comments such as "One thing I think was tough about the class was that because it was a class for non majors, the people in it came from such different backgrounds that it was hard to keep the lessons approachable for complete beginners and interesting for people with experience," which is a noted challenge in designing this course. Some students have significant background in sewing, some have prior knowledge of circuits, and a few have more programming experience than others (although, due to enrollment pressures, the instructor of this course has the option to selectively drop students based on their stated programming expertise). If a version of this course had students with more mixed programming abilities, it might not work as well for those with advanced programming skills as a significant portion of time is dedicated to smaller steps of programming concepts in the beginning of the course.

3.1 Programming Concepts

Despite not having requirements for programming concepts in the final project, a wide range of Arduino keywords are found. In student final projects, nearly all students require proficient use of conditionals, and in some cases, nested conditionals. While loops are not often required in these projects, understanding the built-in Arduino `loop()` function requires conceptual understanding of

loops and functions, without much of the `while`/`for` loop syntax. Very few student final projects use arrays, although their most common use is to keep track of non-consecutive pins that all need the same behavior (this also tends to introduce a need for a `for...` loop to make that behavior happen). Similar adoption can be seen in student assignments of student-created functions, however, Arduino software all uses a built-in `setup()` and `loop()` function, so in some respect, all students are using at least two user-defined functions in their final projects. To encourage deeper use of more complex concepts such as loops, functions, and arrays, it may be necessary to (1) have more than 12-weeks in a semester, and (2) require some sort of state-based programming paradigm with parallel-appearing behaviors or generate a need for verifying inputs. Regardless, these more complex concepts are all exemplified by students in earlier programming assignments in the Electronic Textiles course.

4 Conclusion

In general, the Electronic Textiles course appears to be an enjoyable and worthwhile experience for students, as well as instructors. As this course moves to other institutions, there will be changes in certain logistical parameters, such as programming background of the students as well as length of the semester, that may require shifts in the Electronic Textiles curriculum. The main anticipated challenges (or, alternatively, opportunities for improvement) are: (1) how to better tailor the course to a wider variety of programming abilities, and (2) how to ensure increased spontaneous use of more complex programming concepts. Both of these problems may potentially be solved with the same approach: drastically change the curriculum to focus on students creating a 3-part "collection", thereby integrating the sewing and circuitry more evenly alongside the programming. Adapting a fashion industry-style project portfolio (the "collection") may also change the course culture in a more art-based, expressive way as well. In short, while the Electronic Textiles course in its current, more traditional computer science course organization is successful in educating and exciting students about the creative opportunities opened by computer programming with circuits, these benefits could be even greater if a more creative approach is taken with the course organization as well.

And finally, the Electronic Textiles course is a refreshing option for both instructors and students spending much time considering the use of generative AI. While perhaps AI could generate some of the Arduino code for student projects, the students must have sufficient understanding of how the code connects and interacts with the physical components that they designed in order for their final projects to be successful. While it is not entirely out of the question that inappropriate use of generative AI can occur in the Electronic

Textiles course, the personalized projects, combined with the physical computing aspects, serve as explicit demonstration of student understanding of the programming, digital circuit, and fiber arts concepts from the course.

References

[1] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. "A review of research on parsons problems". In: *Proceedings of the twenty-second australasian computing education conference*. 2020, pp. 195–202.

[2] Iris Howley. "Adapting guided inquiry learning worksheets for emergency remote learning". In: *Information and Learning Sciences* 121.7-8 (2020), pp. 549–557.

[3] Stacey Kuznetsov et al. "Breaking boundaries: strategies for mentoring through textile computing workshops". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2011, pp. 2957–2966.

[4] Sam Lau and Philip Guo. "From" Ban it till we understand it" to" Resistance is futile": How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot". In: *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*. 2023, pp. 106–121.

[5] Rongxin Liu et al. "Teaching CS50 with AI: leveraging generative artificial intelligence in computer science education". In: *Proceedings of the 55th ACM technical symposium on computer science education V. 1*. 2024, pp. 750–756.

[6] Alexandros Merkouris, Konstantinos Chorianopoulos, and Achilles Kameas. "Teaching programming in secondary education through embodied computing platforms: Robotics and wearables". In: *ACM Transactions on Computing Education (TOCE)* 17.2 (2017), pp. 1–22.

[7] Eric Poitras, Brent Crane, and Angela Siegel. "Generative ai in introductory programming instruction: Examining the assistance dilemma with llm-based code generators". In: *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1*. 2024, pp. 186–192.

An Audio Example of GPU Computing *

Justin Douty and David P. Bunde
Computer Science
Knox College
Galesburg, IL 61401
jdouty03@gmail.com, dbunde@knox.edu

Abstract

General-Purpose Graphics Processing Unit (GPGPU) programming is a major paradigm for parallel computing. It takes advantage of the power of modern graphics processors to perform calculations other than those needed for the display. Despite the "General Purpose" in its name, students often associate GPGPU programming with graphics, a tendency that is reinforced by the fact that many common pedadogical examples do use graphics. This paper presents an audio processing example of GPGPU programming, the conversion of a MIDI audio file into the WAV format. This application can be understood without prior background in audio files. We present a laboratory activity based on this example.

1 Introduction

At this point, essentially all computers are parallel computers and it is hard to buy a single-core processor. Even small devices such as cell phones typically have multiple cores. In addition, parallel devices are increasingly characterized by heterogeneity, with cores running at different speeds, special-purpose circuits to optimize specific operations, and increasingly complex types of memory locality [10]. These changes present new challenges and opportunities to programmers and thus to the CS educators who teach them.

One common kind of programmer-visible heterogeneity involves the Graphics Processing Unit (GPU), specialized hardware designed for graphics processing. Graphics operations are characterized by the need to perform the same operation (e.g. a linear transformation) on each of a large number of data items (e.g. every triangle in a large 3D scene). Because each data item needs the same operation, SIMD (single-instruction, multiple-data) operations are appropriate. This, combined with the large number of items involved, led to GPU designs with a large number of cores (100s or 1000s instead of the 10s available on current processors). The availability of this much compute capability, even restricted to perform SIMD operations, led to the idea of General-Purpose GPU (GPGPU) programming, in which the GPU is used for non-graphical calculations.

The most common framework (programming language and runtime environment) for GPGPU programming is CUDA, which is designed and controlled by NVIDIA. CUDA is the focus of this paper, though we believe the content could be translated equally well into other GPGPU programming systems.

A common first example of CUDA programming (e.g. [7]) is to parallelize a simple loop for vector addition, the element-wise addition of array values as follows:

```
for(int i = 0; i < n; i++)
   y[i] = x[i] + y[i];
```

This loop can be easily converted to run on the GPU: Instead of using a loop to repeat the loop body `n` times serially, `n` threads are created and each runs the loop body once to perform the addition for one array cell.

The loop body itself becomes a very short *kernel*, as a function that runs on the graphics card is called. This kernel is invoked from code running on the CPU and it creates a large number of threads, each running the kernel. Each thread uses its identity to determine which loop iteration it is responsible for and performs the necessary computation.

In addition to invoking the kernel to perform the addition itself, a CUDA version of this loop also requires data movement operations because the GPU and CPU have separate memory address spaces. Thus, before the kernel can be invoked, a system call is needed to transfer arrays `x` and `y` from the CPU's address space to the GPU's. After the kernel runs, another call is needed to transfer array `y` back to the CPU's address space.

This vector addition example is useful because it is one of the simplest CUDA programs that does useful work. It illustrates the parts of a CUDA program without a lot of other code. That said, vector addition is hardly an application that excites students. It is also not a good example of CUDA's performance potential; the CUDA vector addition code actually runs slower

than the serial for-loop version because of the time required to move data between the CPU and GPU address spaces.

Simple examples that are more interesting for students can be created from image processing operations, following the idea of Media Computation for teaching CS1 [6]. Bhuchar et al. [2] use operations that can be performed by looping through pixels and adjusting their RGB values, e.g. converting images to grayscale. These operations are easy to understand and the implementations are simple enough for successful completion by students new to CUDA, In addition, because the results are graphical, errors are often readily apparent as visual artifacts.

Unfortunately, the fact that the operations give graphical output also comes with a significant pedagogical drawback: we have found that some students end up thinking that CUDA is mainly useful for graphics. We believe this occurs because we introduce CUDA as running on the GPU and then most of the CUDA programs they see are for graphics applications.

The contribution of this paper is an interesting but relatively simple non-graphical application for CUDA. It comes from audio processing, specifically converting audio data from the MIDI format to the WAV format. This application even retains the ease of error detection we found in graphical applications because audio artifacts are similarly easy to detect.

In this paper, we present the background needed to understand the application and then describe its use as a lab. We also discuss a previous version of the lab which students found confusing as a caution for creating CUDA assignments.

2 Related Work

Before describing our work in detail, we identify some related previous work. Adams et al. [1] present "sonifications", audio-based alternatives to visualizations, for common sorting algorithms to make these algorithms more accessible to the visually impaired. They do not consider CUDA and the audio is added to a separate computation, whereas the goal of our computation is the creation of a WAV file.

The CSinParallel group [9] has several modules for teaching CUDA. One teaches how to time CUDA programs. Another builds on the vector addition example to show how much computation is needed to make CUDA worthwhile. Their "Optimizing CUDA for GPU Architecture" instructor example uses computation of the Mandelbrot set, another graphical example.

Although CUDA is the most common framework for teaching GPGPU programming, other approaches exist. Fuentes et al. [5] describe a heterogeneous computing course using DPC++.

Others have taught about heterogeneous computing systems other than GPGPUs and CUDA. Carloni et al. [3] and Frachtenberg [4] describe undergraduate elective courses on heterogeneous computing that take a hardware design approach.

3 Audio Formats and Conversion Process

Now we present background on the audio file formats used in our application and the conversion process. The first audio format we use is the WAV format. An audio file in WAV format is a header followed by a long list of music intensities, one per time unit of the song. Playing a song in this format is straightforward; the program simply goes through this list, playing the appropriate intensity at each unit of time. This format is exactly what is visualized when sounds are shown as a waveform.

The second audio format we use is MIDI. This format is more like sheet music, telling when to play each instrument. Specifically, the file contains a list of "notes", each identifying an instrument and giving the start time and duration of the note. In order to play a MIDI file, a program also needs the sound corresponding to each instrument, which is stored separately in a WAV-like format giving the intensity per time unit. An advantage of the MIDI format is that it generally yields a shorter file since its length is proportional to the number of notes rather than the number of time units in the music.

To transform a MIDI file into a WAV one (or to play it), each note must be processed to create the sound it contributes at each time unit and these contributions must be added together; multiple notes from the same or different instruments can play simultaneously. Figure 1 contains the code for adding a single note to `audiodata`, the array containing the entire song's waveform. This code is run for each note, adding them to the final waveform one at a time.

The first loop in Figure 1 adds the main part of the note, when it is actively played (from time `start` to time `end`). The second loop accounts for the note's decay as it fades away after being played (from time `end` to time `decayEnd`). The waveform of the instrument to play is stored in the array `instSection.data`. The length of this waveform is scaled based on the value of `pitchM` and its height is scaled based on the value of `velocityM` to create different notes from that instrument.

Each of the loops in Figure 1 performs an operation analogous to vector addition, with the waveform of a note being added to the song's waveform. The operations in these loops are more complicated than standard vector addition because of the following differences:

- the vectors have different lengths, with the song's waveform array having

```
/* Render Note */
for(unsigned long x = start; x < end; x++) {
   unsigned long offset = (x - start) * pitchM;
   audioData[x] += instr.data[offset] * velocityM;
}

/* Render Note Decay */
unsigned long size = decayEnd - end;
for(unsigned long x = end; x < decayEnd; x++) {
   unsigned long offset = (x - start) * pitchM;
   float relVol = (decayEnd - x) / (float) size;
   audioData[x] += instr.data[offset] * velocityM * relVol;
}
```

Figure 1: Code to add a single note to the song waveform in `audiodata`.

more cells than the note's,

- the vectors start at different index values, with the note being added to the song's waveform starting at an index other than 0, and
- the loops scale the note vector's length and its values, rather than simply adding the values of one array to the other.

Even so, one way to think of the format conversion is as a better-motivated, more-complicated version of vector addition.

Another advantage of the format conversion application is that it requires less data movement; the instrument sounds are the only data that is transferred to the GPU and they are reused by multiple notes. This reduction in data movement makes it easier to achieve a performance improvement over the serial version.

4 Lab Assignment

Now we discuss the use of this application in a lab assignment. It has been used since 2023 in offerings of Introduction to Computing Systems, a course which presents systems issues with a focus on what every programmer needs to understand in order to write high-performance code. The students in this class learn C and assembly so they can understand how code actually runs on the computer. The course also covers caching, parallelism/concurrency, binary data representation, and basic concepts of networking and security. It uses the

Dive into Systems textbook [8] and is typically taken in the second year of the Computer Science major.

Lab assignments in this course are mainly time for the students to practice course concepts. Students are given a handout stepping them through the activity while the instructor provides help as needed. Lab assignments are not graded directly, but are similar to graded homework assignments students receive after lab.

Each time it was assigned, this lab was given after CUDA was introduced in class after the completion of a lab based on image manipulation tasks such as converting an image to grayscale, blurring the image, etc.

In this lab, the students begin by reexamining and modifying the CUDA program to add vectors. Specifically, they are asked to modify it so that a shorter array is added to a longer array starting at a cell other than the first. This is exactly the version of the problem that they need to solve to add a note to `audioData`, but they solve it first with arrays of integers (one all 0s and one with value `i` in cell `i`). They are also specifically told to solve it using one thread per cell of the shorter array.

After that introductory task, the students switch to creating the WAV file. They are given a working serial version and an incomplete CUDA program that sets up the computation, performs all the needed data transfers, and invokes the CUDA kernel. The kernel is meant to be a translation of the code in Figure 1, which is included in the lab handout to specifically highlight it.

With this process, many students were able to finish the lab during the lab period and nearly all completed it when it was subsequently assigned as homework.

5 Discussion

We believe it is instructive to also consider a slight variation of the lab, one that didn't work as well. In that version, rather than explicitly referring back to the vector addition problem and solving the variation with different length vectors, the students were asked to solve the WAV problem directly in two steps. In the first version, they are asked to use `decayEnd` threads, one for each unit of time from 0 until the note has completely faded. The thread numbers correspond to the values of `x` in Figure 1. The students need the kernel to run the appropriate loop body depending on the value of `x`: do nothing if `x < start`, run the first loop body if `start` $\leq$ `x` $<$ `end`, and run the second loop body if `x` $>$ `end`. Then, in a second version, they are asked to reduce the number of threads by removing those corresponding to times before the note starts. This is essentially asking them to solve the problem of adding vectors of different lengths.

As mentioned above, this version of the lab was much less successful. The

students struggled with the basic setup; without an explicit reference back to the vector addition problem, many of them initially tried to include loops in their kernel, missing the key idea of replacing loop iterations with threads. Even when this was clarified, they still struggled compared to students who did the different length vector problem first. The problems are analogous, but having the extra code (the contents of figure 1 instead of an array) obscured the structure of the problem from them. Students were also frustrated because the final solution was so short, but they had struggled so much to find it.

Essentially this episode is a reminder of the importance of providing students with adequate scaffolding, particularly when they are grappling with new concepts, like SIMD execution replacing loop iterations. This seems obvious in retrospect, but wasn't when designing the initial version of the lab or helping students complete it.

Going forward, we see two ways to use this application. One is to continue to use it as a lab; based on informal student feedback, we consider the current version of the lab to be successful. Converting music files is an application many of them find interesting (unlike vector addition) and that it is non-graphical helps broaden their idea of GPGPU programming. We also believe one could use this example in lecture; introduce the vector addition example, work through the variation with different length vectors with them, and then present this application and its solution. This approach can be used to either reduce coverage of CUDA (relative to the two-lab version) or to make room for a different second CUDA assignment.

Acknowledgements

This work was partially supported by the National Science Foundation through award OAC-1829554. We also thank the anonymous referees, who provided helpful comments on an earlier version of this manuscript.

References

[1] J.C. Adams et al. "The Sounds of Sorting Algorithms: Sonification as a Pedagogical Tool". In: *Proc. 53rd ACM Technical Symp. Computer Science Education (SIGCSE)*. Vol. 1. 2022, pp. 189–195.

[2] K. Bhuchar et al. "ToUCH Module D1: Introduction to CUDA Programming". URL: `https://github.com/TeachingUndergradsCHC/modules/tree/master/Programming/cuda` (visited on 08/01/2026).

[3] L.P. Carloni et al. "Teaching heterogeneous computing with system-level design methods". In: *Proc. Workshop Computer Architecture Education*. 2019, pp. 1–8.

[4] E. Frachtenberg. “Experience and practice teaching an undergraduate course on diverse heterogeneous architectures”. In: *Proc. Workshop on Education for High-Performance Computing (EduHPC)*. 2021.

[5] J. Fuentes, D. López, and S. González. “Teaching heterogeneous computing using DPC++”. In: *Proc. Workshop on Education for High-Performance Computing (EduHPC)*. 2022.

[6] M. Guzdial. “Exploring hypotheses about media computation”. In: *Proc. 9th Ann. ACM Conf. Intern. Computing Education Research (ICER)*. 2013, pp. 19–26.

[7] M. Harris. “An Even Easier Introduction to CUDA”. May 2025. URL: `https://developer.nvidia.com/blog/even-easier-introduction-cuda/` (visited on 08/01/2026).

[8] S.J. Matthews, T. Newhall, and K.C. Webb. “Dive into Systems”. URL: `https://diveintosystems.org/` (visited on 08/01/2026).

[9] S.J. Matthews et al. *CSinParallel Project*. URL: `http://csinparallel.org/` (visited on 08/01/2026).

[10] M. Zahran. “Heterogeneous computing: Here to stay”. In: *CACM* 60.3 (2017), pp. 42–45.

LLM Guided Computational Exploration of Open Problems in Graph Labeling*

Adithya Kulkarni and Jay Bagga
Department of Computer Science
Ball State University
Muncie, IN 47306
{adithya.kulkarni, jbagga}@bsu.edu

Abstract

Open problems in graph labeling provide rich opportunities for computational exploration, but students often struggle to enter the literature, identify relevant structures, and translate conjectures into testable experiments. This paper studies how Large Language Models can support these early stages of mathematical research when used as guided exploratory tools rather than autonomous theorem provers.

We examine LLM-guided exploration in two graph labeling settings: graceful line graphs and distance magic graphs. Using web versions of ChatGPT 5.5 Thinking, Gemini 3.1 Thinking and Pro, and Claude Sonnet 4.6 Adaptive, we analyze model responses to solved and open problems involving line graph gracefulness, distance magic graphs of order 11, and magic constants realized by connected distance magic graphs. Although the prompting structure was adapted from prior work, the contribution of this paper is pedagogical: a verification-centered workflow for orientation, motif extraction, computational translation, and validation.

The results show that LLMs can help identify definitions, relevant graph families, parity constraints, neighborhood-sum equations, and computational approaches such as graph enumeration, constraint satisfaction, and integer programming. Because proof attempts may contain unsupported reasoning or incomplete arguments, LLM outputs must

be carefully validated. We therefore propose a framework in which LLMs serve as initialization tools for computational research, while rigorous proof and critical evaluation remain central learning goals. The prompts and experimental outputs are available at `https://github.com/kulkarniadithya/Dual_Role_of_LLM`.

1 Introduction

Graph labeling problems provide a rich setting for computational research in computer science education. They are often easy to state but difficult to solve because they combine discrete structures, arithmetic constraints, graph transformations, computational search, and proof. Graceful labeling, distance magic labeling, and related variants have generated extensive literature and many open problems [6]. These problems are valuable in advanced undergraduate and graduate courses because they connect graph theory, algorithms, combinatorial experimentation, and mathematical reasoning.

Students entering this type of research often face a difficult starting point. They may not know which definitions are central, which graph families are relevant, or how to translate a conjecture into a computational experiment. LLMs can support conceptual clarification and exploratory problem solving in educational settings [14, 16]. At the same time, students may over trust fluent but incorrect model output, especially in STEM contexts where verification requires domain expertise [9, 12]. These concerns are reinforced by studies documenting hallucination and failures in mathematical reasoning by LLMs [8, 1, 5, 7].

This paper examines a limited but pedagogically useful role for LLMs. Rather than asking whether they can solve open mathematical problems, we ask whether they can help students and researchers begin such investigations responsibly. We study LLMs as research scaffolding tools that may help organize literature, identify relevant graph families, surface recurring structural motifs, and suggest computational search strategies. In this framing, model output is not treated as mathematical authority; it is treated as a starting point for verification, computation, and critique.

The study focuses on two graph labeling domains: graceful line graphs and distance magic graphs. In the first setting, the line graph $L(G)$ of a graph G has one vertex for each edge of G, with adjacency determined by edge incidence in G. This setting is useful because graceful labelings of G label the original vertices, whereas graceful labelings of $L(G)$ label objects corresponding to edges of G, creating a natural interaction between labeling constraints and graph transformations [3, 4]. In the second setting, a distance magic labeling assigns labels bijectively to vertices so that every vertex has the same open-neighborhood sum, called the magic constant. Recent work has characterized

magic constants arising from distance magic graphs and shown the difficulty of constructing such graphs under prescribed conditions [11, 2].

This paper builds on prior work evaluating LLMs on solved and unsolved graph theory problems [10], but shifts the focus from proof correctness on a single problem pair to pedagogical workflow design across multiple graph labeling domains. We evaluate web versions of ChatGPT 5.5 Thinking, Gemini 3.1 Thinking and Pro, and Claude Sonnet 4.6 Adaptive on four tasks: a solved graceful line graph problem, an open graceful line graph problem, a solved distance magic construction task of order 11, and an open problem concerning which magic constants are realized by connected distance magic graphs.

The central contribution is a framework for transforming LLM generated mathematical exploration into verifiable computational research activity. The framework has four stages: orientation, motif extraction, computational translation, and verification. Through this framework, LLMs can help students understand problem settings, identify reusable structural ideas, convert those ideas into computational experiments, and check claims against sources, examples, computation, and expert reasoning. The goal is not to replace proof, but to support research based learning while preserving rigorous verification as a central outcome.

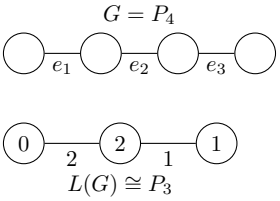
2 Background and Related Work

This section reviews the literature needed to frame the study: graph labeling as a computational research domain, the educational use of Large Language Models, and the need for verification when LLMs are used in mathematically rigorous settings.

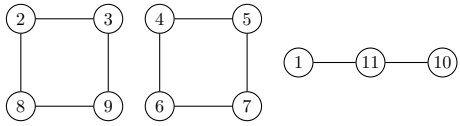
2.1 Graph Labeling and Computational Exploration

Graph labeling is a broad area of graph theory in which integers are assigned to vertices, edges, or both under arithmetic constraints. Many labeling problems are easy to state but difficult to solve because local labeling rules must satisfy global combinatorial conditions. Gallian's dynamic survey documents the breadth of this area, including graceful, magic, antimagic, harmonious, and related labeling variants [6]. A graph with q edges is graceful if its vertices can be assigned distinct labels from $\{0, 1, \ldots, q\}$ so that the induced edge differences are exactly $\{1, 2, \ldots, q\}$ [6]. The persistence of major open problems, including the Graceful Tree Conjecture, illustrates the difficulty of moving from examples and computational evidence to general proof [6].

For the present study, the line graph setting is important because the graph operation changes the objects being labeled. A graceful labeling of G assigns labels to vertices of G, whereas a graceful labeling of $L(G)$ assigns labels to



(a) A graceful labeling of a line graph.



(b) A distance magic labeling of $P_3 \cup 2C_4$ with $k = 11$.

Figure 1: Examples of the two labeling settings used in this study.

vertices corresponding to edges of G. This shift is a source of both mathematical difficulty and computational interest, since it connects graceful labeling to graph transformations, parity obstructions, clique formation, and inverse reconstruction. Figure 1(a) gives a small example: if $G = P_4$, then $L(G) \cong P_3$, and the displayed labeling of $L(G)$ has induced edge differences $\{1, 2\}$.

The distance magic setting provides a complementary labeling constraint. A graph of order n is distance magic if its vertices can be labeled bijectively with $\{1, 2, \ldots, n\}$ so that every vertex has the same open-neighborhood sum, called the magic constant [6, 2, 11]. The condition can be expressed as

$$Af = k\mathbf{1},$$

where A is the adjacency matrix, f is the labeling vector, and $\mathbf{1}$ is the all-ones vector. Figure 1(b) shows a distance magic labeling of $P_3 \cup 2C_4$ with magic constant 11. Together, these examples illustrate the two types of structure studied in this paper: label differences under graph transformations and neighborhood-sum constraints.

2.2 LLMs in Education and Research Workflows

LLMs are increasingly used in education for conceptual clarification, programming support, assignment assistance, and literature exploration. Recent work reports potential benefits for personalized learning and instructional support while emphasizing concerns about reliability, transparency, and responsible integration [14, 16]. In computing education, these tools are especially visible because they can explain code, generate examples, decompose problems, and summarize unfamiliar material.

Advanced mathematical contexts introduce additional risk. Students may ask LLMs to interpret definitions, summarize papers, suggest proof ideas, or generate computational approaches before they have the expertise needed to

evaluate correctness. Prior work in STEM education shows that students may accept plausible but incorrect AI-generated explanations, particularly when the output is fluent and confident [9, 12]. This concern is directly relevant to graph labeling, where an argument may appear reasonable while failing because of a missing hypothesis, invalid implication, or unsupported arithmetic step.

The educational value of LLMs should therefore not be measured only by whether they produce final answers. In research-based learning, students often need help entering a domain, locating terminology, identifying relevant graph families, and translating broad conjectures into computational tasks. Used carefully, LLMs can serve as orientation tools that help students begin this process. The central pedagogical challenge is to ensure that such outputs remain provisional and are followed by verification.

2.3 LLM Reasoning Limits and Verification

A substantial body of work documents hallucination and reasoning failures in LLMs. Recent surveys describe hallucination mechanisms and mitigation challenges, and studies of mathematical reasoning document unsupported deductions, incorrect symbolic manipulation, hallucinated problem features, and failures to identify errors in generated arguments [8, 1, 5, 7, 15]. These risks are especially important when LLM outputs include theorem names, author attributions, or claims about open problems.

Mitigation methods can improve reliability in some settings, but they do not remove the need for human judgment in open mathematical domains. Work on AI-assisted mathematics also reinforces a useful educational principle: machine-generated suggestions are most valuable when paired with human interpretation, computational testing, and proof verification [13]. The present study applies that principle to graph labeling education by treating LLMs as tools for orientation, motif extraction, computational translation, and critique rather than as autonomous theorem provers.

3 LLM Guided Computational Research Workflow

This study uses a qualitative comparative case study design to examine how LLMs can support exploratory computational research in graph labeling. The goal is not to benchmark models or evaluate prompt engineering, but to study how model responses can help students and researchers move from unfamiliar literature toward structured computational inquiry.

3.1 Experimental Setting

The study was organized around two graph labeling domains: graceful line graphs and distance magic graphs. Each domain included one solved task and one open problem. The solved tasks served as calibration cases for determining whether the models could recover established reasoning or known constructions. The open problems tested whether the models could identify relevant structures, partial approaches, and computational experiments without claiming unsupported solutions.

For graceful line graphs, the solved problem asked whether nongracefulness of G implies gracefulness of $L(G)$; this is false, as cycles C_n with $n \equiv 1$ or $2 \pmod 4$ are nongraceful and satisfy $L(C_n) \cong C_n$. The open problem asked whether gracefulness of $L(G)$ implies gracefulness of G. For distance magic graphs, the solved task asked for distance magic graphs of order 11, and the open problem asked which magic constants are realized by connected distance magic graphs.

The experiments used web based versions of ChatGPT 5.5 Thinking, Gemini 3.1 Thinking, Gemini 3.1 Pro, and Claude Sonnet 4.6 Adaptive. Web interfaces were used because they reflect common student practice: users upload papers, pose mathematical questions, and ask follow up questions interactively. No model was accessed through an API, fine tuned, or modified with custom system instructions. Each model was evaluated in separate conversations for each problem domain, and all outputs were saved for qualitative analysis.

3.2 Materials and Protocol

For the graceful line graph experiments, the models were provided with *A Dynamic Survey of Graph Labeling* [6] and *On Graceful Line Graphs.*[1] For the distance magic experiments, the models were provided with *A Dynamic Survey of Graph Labeling* [6], Pawar et al.'s characterization of magic constants arising from distance magic graphs [11], and Arumugam, Kamatchi, and Vijayakumar's work on uniqueness of D-vertex magic constants [2].

The prompting protocol was adapted from prior work on solved and unsolved graph theory problems [10]. To make the present study self-contained, we summarize the protocol here. Each interaction followed an eight-stage sequence: (1) restate the problem, identify key definitions, and explain why the problem is challenging; (2) brainstorm possible approaches; (3) identify related areas of graph theory or graph labeling; (4) list relevant known results or conjectures, with uncertainty explicitly noted; (5) propose high-level strategies and anticipated obstacles; (6) attempt a proof or disproof; (7) critique the

[1]This document was obtained from the authors at the 2025 CCSC Midwest Conference and was used as background material for definitions and context.

proof attempt by identifying incorrect or incomplete steps; and (8) revise the argument in light of that critique. Several prompts explicitly instructed the models not to invent theorems, authors, or unsupported results. The protocol was used as an instrument for eliciting model behavior, not as a contribution in itself.

3.3 Analysis and Expert Review

Outputs were analyzed through four functions. *Orientation* refers to whether the model clarified definitions, problem status, relevant graph families, and known constraints. *Motif extraction* refers to whether the model identified reusable structures such as parity obstruction, fixed point behavior of cycles, clique formation, inverse reconstruction, neighborhood sum equations, regularity, graph joins, or connectedization. *Computational translation* refers to whether the model converted mathematical observations into testable procedures such as graph enumeration, backtracking, constraint satisfaction, integer programming, degree filtering, or matrix formulations such as $Af = k\mathbf{1}$. *Verification* refers to whether claims could be checked against sources, examples, computation, or expert reasoning.

All outputs were reviewed qualitatively by a graph theory expert with experience in line graphs and graph labeling. The expert evaluated mathematical correctness, relevance of proposed graph families, accuracy of theorem statements, validity of deductions, usefulness of computational suggestions, and whether the models distinguished solved results from open questions. The goal was not to rank models numerically, but to understand how model responses can support or mislead students during exploratory research. A response was treated as useful when it clarified the problem, suggested a verifiable direction, or exposed an obstacle worth investigating; a proof attempt was not treated as valid unless independently justified.

4 Experiments and Results

This section summarizes findings from the LLM guided exploration of four graph labeling problems. The results are organized by the role LLMs played in supporting computational research rather than by individual prompts. Across both graph labeling domains, the models were most useful for orienting the user, surfacing structural motifs, and translating those motifs into computational tasks. They were less reliable when asked to produce final proofs or justify broad claims without verification.

The experiments involved two domains. In the graceful line graph setting, the models examined one solved problem, whether nongracefulness of G implies gracefulness of $L(G)$, and one open problem, whether gracefulness of $L(G)$

implies gracefulness of G. In the distance magic setting, the models examined one solved construction task, finding distance magic graphs of order 11, and one open characterization problem, determining which magic constants are realized by connected distance magic graphs.

4.1 Orientation and Calibration

The solved problems served as calibration tasks. In the graceful line graph experiment, the models generally identified cycles as the decisive counterexample family. Since $L(C_n) \cong C_n$, a nongraceful cycle remains nongraceful under the line graph operation. Thus, for $n \equiv 1$ or $2 \pmod 4$, the cycle C_n gives a counterexample to the claim that nongracefulness of G forces gracefulness of $L(G)$; C_5 is the simplest example.

In the distance magic experiment, the models frequently identified $P_3 \cup 2C_4$ as a distance magic graph of order 11 with magic constant $k = 11$, as shown in Figure 1(b). This construction is explicit and easy to check: the P_3 component can be labeled so that the endpoint labels sum to 11, and each C_4 component can be labeled using opposite pairs that produce the same neighborhood sum. These calibration cases show that LLMs can recover useful examples from established material, but they also illustrate why verification remains necessary. A correct example is useful for learning; it does not make the model a reliable theorem prover.

4.2 Motif Extraction and Computational Translation

The open problems showed the models' strongest educational value. They did not solve the open problems, but they produced useful exploratory structure. For the open graceful line graph problem, the models repeatedly identified the difficulty of moving from a graceful labeling of $L(G)$ to a graceful labeling of G. A labeling of $L(G)$ assigns values to vertices corresponding to edges of G, while a graceful labeling of G assigns values to the original vertices. The models connected this mismatch to inverse line graph structure, parity obstruction, clique formation, and Whitney type reconstruction.

For the open connected distance magic problem, the models emphasized neighborhood sum equations, regularity, complete multipartite graphs, graph joins, circulant graphs, fractional total domination, and connectedness constraints. The equation

$$Af = k\mathbf{1}$$

emerged as a central computational representation, where A is the adjacency matrix, f is the labeling vector, and $\mathbf{1}$ is the all-ones vector. This formulation translates the labeling condition into a linear algebraic and combinatorial search problem.

The models also translated these motifs into computational tasks. For graceful line graphs, they suggested generating small graph families, computing line graphs, applying parity filters, and testing graceful labelings through backtracking or constraint based search. For distance magic graphs, they suggested neighborhood subset search, exact cover formulations, integer programming, adjacency matrix filtering, and connectedness checks. These were not complete algorithms, but they gave students concrete pathways from a mathematical question to an implementable experiment.

Not every model-generated direction was actionable. Some suggestions were too generic to guide research, such as naming broad graph families without connecting them to the labeling constraints, restating the search problem without specifying variables or constraints, or citing theorem-level ideas that required source checking. We therefore treated LLM output as a pool of candidate directions rather than as research contributions. A suggestion was useful only when it could be made precise, tested computationally, or connected to a verifiable structural obstacle. Thus, the models expanded the space of possible approaches, but expert review was needed to separate promising directions from irrelevant or underspecified ones.

4.3 Verification and Failure Modes

The experiments also showed clear limits. The most common failure mode was incomplete proof reasoning: models sometimes moved from a plausible observation to a general claim without necessary hypotheses or justification. A second issue was imprecise theorem recall. Even when instructed not to invent results, models sometimes produced statements that required source checking. A third issue was overgeneralization from examples, especially when a correct analysis of cycles, regular graphs, or complete multipartite graphs was treated as evidence for a broader claim.

Qualitative differences among the models were observed, but they do not change the main conclusion. ChatGPT 5.5 Thinking tended to produce structured explanations with explicit attention to uncertainty. Gemini 3.1 Thinking and Pro often generated broad sets of approaches and computational strategies. Claude Sonnet 4.6 Adaptive frequently produced detailed mathematical narratives and identified structural obstacles. All models, however, required verification.

Overall, LLMs were most useful as research scaffolds. They helped orient students to unfamiliar literature, surface structural motifs, and suggest computational experiments. They were least reliable when asked to produce final proofs. The appropriate learning goal is therefore not to have students accept LLM output, but to teach them how to turn that output into verifiable computation, careful reading, and rigorous mathematical reasoning.

5 Educational Implications

The results suggest three uses of LLMs in computing education: onboarding, computational translation, and critique. Graph labeling is a useful setting for these uses because students must connect definitions, graph families, algorithmic search, and proof. In this context, the LLM should be framed as a tool for initiating inquiry rather than as a source of mathematical authority.

First, LLMs can serve as research onboarding tools. Students beginning work in graph labeling often do not know which definitions, examples, or prior results matter. In the experiments, the models helped identify starting points such as cycles, complete graphs, regular graphs, graph joins, and neighborhood-sum equations. Such responses can help students locate terminology and form initial search strategies, but they must be verified against authoritative sources before being used. Second, LLMs can help translate mathematical ideas into computational tasks. For graceful line graphs, model outputs suggested generating graph families, computing line graphs, testing graceful labelings, and using parity or degree constraints as filters. For distance magic graphs, they suggested adjacency matrix formulations, connectedness checks, exact cover models, and integer programming approaches. These suggestions create assignments in which students implement, test, and evaluate a computational direction rather than simply report an LLM response. Third, LLM output can be used as an object of critique. The models sometimes produced incomplete arguments, overgeneralized from examples, or stated claims requiring source checking. Instructors can use such failures to teach verification by asking students to separate definitions, examples, known results, conjectures, and unsupported claims, then test selected claims through computation, literature review, and proof.

Together, these observations support a verification-centered classroom workflow. Students use an LLM to orient themselves to a problem, identify structural motifs, translate one motif into a computational experiment, and verify the result through sources, examples, code, and proof. In this workflow, the LLM helps initialize inquiry, but the student remains responsible for mathematical judgment.

6 Limitations and Future Work

This qualitative study should be interpreted as support for pedagogical workflow design rather than as a benchmark of model performance or a classroom intervention. The use of web-based LLMs reflects realistic student and instructor practice, but limits reproducibility because model behavior may change over time. The study is also limited to two graph labeling domains, graceful

line graphs and distance magic graphs, with qualitative expert evaluation. Because no undergraduate research group implemented the workflow, the paper does not claim demonstrated learning gains or completed research outcomes; instead, it proposes a verification-centered workflow to be tested in future classroom and mentoring settings.

Future work will test the proposed workflow with undergraduate research students and in advanced computing courses. Such studies should examine whether LLM-guided exploration helps students read mathematical literature, formulate research questions, design computational experiments, and critique AI-generated reasoning. Future work should also implement selected strategies suggested by the models, including graph enumeration, constraint satisfaction, integer programming, and adjacency matrix methods. Finally, a larger corpus of graph labeling problems and proofs could be developed to study recurring motifs such as parity, clique formation, regularity, connectedness, and neighborhood-sum balance. In the long term, these motifs may support a compositional view of graph labeling discovery through transformations $T : (G, \ell) \mapsto (G', \ell')$. In all cases, LLM assistance must remain paired with proof, computation, and expert verification.

7 Conclusion

This paper examined how LLMs can support exploratory computational research in graph labeling. Across graceful line graph and distance magic graph case studies, the models were useful for orientation, motif extraction, and computational translation. They helped identify relevant definitions, graph families, structural constraints, and possible computational approaches.

The main contribution is a verification-centered framework in which LLMs help initialize research by organizing literature, surfacing structural motifs, and suggesting computational experiments. The framework is intentionally limited: model-generated claims must be checked against source materials, examples, computation, and proof. This approach offers a responsible path for integrating LLMs into graph theory, algorithms, artificial intelligence, and undergraduate research mentoring, while leaving classroom validation and student research outcomes as important future work.

References

[1] Janice Ahn et al. “Large language models for mathematical reasoning: Progresses and challenges”. In: *arXiv preprint arXiv:2402.00157* (2024).

[2] Subramanian Arumugam, Nainarraj Kamatchi, and Gurusamy Rengasamy Vijayakumar. “On the uniqueness of D-vertex magic constant”. In: *Discussiones mathematicae graph theory* 34.2 (2014), pp. 279–286.

[3] Jay Bagga. “Old and new generalizations of line graphs”. In: *International Journal of Mathematics and Mathematical Sciences* 2004.29 (2004), pp. 1509–1521.

[4] Jay Bagga and Lowell Beineke. “A survey of line digraphs and generalizations”. In: *Discrete Math. Lett* 6 (2021), pp. 68–83.

[5] Johan Boye and Birger Moell. “Large language models and mathematical reasoning failures”. In: *arXiv preprint arXiv:2502.11574* (2025).

[6] Joseph A Gallian. “A dynamic survey of graph labeling”. In: *Electronic Journal of combinatorics* 1.DynamicSurveys (2018), DS6.

[7] Alex Heyman and Joel Zylberberg. “Reasoning large language model errors arise from hallucinating critical problem features”. In: *arXiv preprint arXiv:2505.12151* (2025).

[8] Lei Huang et al. “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions”. In: *ACM Transactions on Information Systems* 43.2 (2025), pp. 1–55.

[9] Lars Krupp et al. “Unreflected acceptance–investigating the negative consequences of ChatGPT-assisted problem solving in physics education”. In: *arXiv preprint arXiv:2309.03087* (2023).

[10] Adithya Kulkarni, Mohna Chakraborty, and Jay Bagga. “Evaluating Large Language Models on Solved and Unsolved Problems in Graph Theory: Implications for Computing Education”. In: *J. Comput. Sci. Coll.* 41.9 (May 2026), pp. 83–100. ISSN: 1937-4771. DOI: 10.1145/3806817.3806823. URL: https://doi.org/10.1145/3806817.3806823.

[11] Ravindra Pawar et al. “A complete characterization of magic constants arising from distance magic graphs”. In: *Journal of Combinatorial Mathematics and Combinatorial Computing* 123 (2024).

[12] Marlene Steinbach et al. “When LLMs Hallucinate: Examining the Effects of Erroneous Feedback in Math Tutoring Systems”. In: *Proceedings of the Twelfth ACM Conference on Learning@ Scale*. 2025, pp. 139–150.

[13] Trieu H Trinh et al. “Solving olympiad geometry without human demonstrations”. In: *Nature* 625.7995 (2024), pp. 476–482.

[14] Shen Wang et al. “Large language models for education: A survey and outlook”. In: *arXiv preprint arXiv:2403.18105* (2024).

[15] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. “Hallucination is inevitable: An innate limitation of large language models”. In: *arXiv preprint arXiv:2401.11817* (2024).

[16] Lixiang Yan et al. “Practical and ethical challenges of large language models in education: A systematic scoping review”. In: *British Journal of Educational Technology* 55.1 (2024), pp. 90–112.

Database Design Review Game*

Nifty Assignment

James Vanderhyde
Computer Science
Saint Xavier University
Chicago, IL 60655
vanderhyde@sxu.edu

This assignment is an exercise for a course on relational databases. It is a group activity to practice database conceptual and logical modeling. I use it on the last day of class, as review for the final exam. After this assignment, exam performance on the database modeling questions improved.

The format of the activity is based on the children's game Telephone, in which the players whisper a message around a circle to see how much the message is modified by the end. (There is also a similar game called *Telestrations*, published by USAopoly in 2009.) In this activity, the class is split into groups of four. Each group gets four packets. The front of each packet is a set of requirements for a database. The first person creates an entity-relationship diagram based on the requirements. Then they fold the page with the requirements over so it can't be seen and pass the ER diagram on to another group member. At the same time, they receive an ER diagram from someone else in the group. Phase 2 involves converting the ER diagram to a relational database schema. Packets are exchanged again. In Phase 3, students reverse-engineer the schema back into an ER diagram. Finally, in Phase 4, the ER diagram is interpreted back into a list of requirements. At the conclusion, the students can look at the requirements at the beginning to see how well the whole process communicated the requirements.

For this activity, you need four sets of database requirements that the students have not seen previously in the course. I found some in old databases textbooks. You need one 5-page packet for each person. The packets are the same except for the first page, which has the requirements and a line blocking out the rest of the page.

Configuring a Client-Server Network Contains Two Servers With Two VLANs on the Same Layer 2 Switch Using Cisco Packet Tracer *

Nifty Assignment

Imad Al Saeed
Computer Sciences Department
Saint Xavier University
Chicago, IL 60655
alsaeed@sxu.edu

This assignment is designed for an Introduction to Networks course intended for students who have little or no previous knowledge of computer networking. The purpose of this activity is to help students understand the basic concepts of layer 2 of the OSI model through hands-on practice. In this assignment, the instructor demonstrates how to build a simple Client-Server network using two servers and multiple end devices connected to the same layer2 switch. Students will learn how to configure the servers so they can automatically assign IP addresses dynamically to end devices connected to the switch. Students will also configure a network switch to divide devices into separate groups on the same physical switch. These groups are called Virtual Local Area Networks (VLANs). Devices that belong to the same VLAN will be able to communicate and ping each other successfully. However, devices located in different VLANs will not be able to communicate with one another unless additional routing is configured. This exercise helps students understand how networks can be segmented for better organization, security, and traffic management. To complete this assignment, students must use a networking simulation software called Cisco Packet Tracer. The software can be downloaded free of charge from the Cisco Academy (Netadac.com) website and installed on the student's computer. Students are required to use the latest version of the software to

ensure compatibility with the networking features used in this assignment. The assignment should be completed by following the steps provided below:

- **Step 1:** Build a client–Server network using one switch of type 2960, two servers. and four laptop computers.
- **Step 2:** Configure the switch with two separate VLANs.
- **Step 3:** Place the first server and two laptops in the first VLAN group.
- **Step 4:** Place the second server and the remaining two laptops in the second VLAN group.
- **Step 5:** Assign the IP address **70.0.0.1** to the first server and configure it as the DHCP services for the first VLAN.
- **Step 6:** Assign the IP address **80.0.0.1** to the second server and configure it as the DHCP services for the second VLAN.
- **Step 7:** Make sure that the two laptops in the first VLAN receive IP addresses automatically from the server with IP address **70.0.0.1**.
- **Step 9:** Make sure that the two laptops in the second VLAN receive IP addresses automatically from the server with IP address **80.0.0.1**.
- **Step 10:** Test network connectivity within each VLAN by pigging your PCs. Pinging PCs on the same VLAN should respond back, while pinging PCs on different VLAN should respond with "Request time Out" to ensure all devices can communicate correctly with their assigned server.

What Does a Scheduler Sound Like? Proving CS Mastery in Any Medium*

Nifty Assignment

Benjamin T. Fine
Department of Computer Science
University of Wisconsin - Eau Claire
Eau Claire, WI 54701
finebt@uwec.edu

Most computer science assignments ask students to write code, prove theorems, or analyze algorithms. This assignment invites something different: demonstrate, explain, or teach a course concept through a creative medium of the student's choosing (e.g., painting, music, performance, literature, comics, even crochet). It has been used as an elective in courses across the computer science and artificial intelligence curricula.

The structure is open but tightly scaffolded. Before beginning, each student pitches an idea to the instructor. Together they workshop it and reach a "handshake agreement" on how the work will connect to the technical content and what C, B, and A grades look like for that specific project. Approval is recommended before students start but is not required. Crucially, the instructor never grades artistic skill, only the student's ability to connect, display, and communicate course content through a different medium.

The results from this assignment surface conceptual understanding that traditional assessments rarely reveal. In operating systems, a student composed music in which multiple themes broke apart according to process scheduling concepts. In an AI course, a student designed the UX for a story-development tool, foregrounding interaction design rather than algorithms. In data structures and algorithms, students rendered Big-O notation and complexity analysis through distinct crochet patterns. Others built comic-book characters that embodied a single concept.

This assignment easily works across tech-focused curricula. From introductory courses to upper- division electives, it reaches students whose strengths

lie outside conventional programming. It rewards genuine understanding over polish, invites real student agency, and consistently produces work students are proud to share.

A Three-Project Series for Building a Virtual Internetwork*

Nifty Assignment

Xin Sun
Computer Science Department
Ball State University
Munice, IN 47304
xsun6@bsu.edu

We present a series of three scaffolded programming projects where undergraduate students build a distributed virtual internetwork from scratch. Rather than interacting with a passive simulation, students implement core networking algorithms directly, including the Ethernet learning switch algorithm (Project 1), hop-by-hop IP forwarding (Project 2), and dynamic routing protocols like Distance Vector (Project 3), by creating standalone host, switch, and router socket programs. Additionally, students design a configuration file format (and parser) to encode the network topology.

A key strength of this assignment suite is its deliberate pedagogical design, which keeps the experience both realistic and manageable. (1) Each project directly builds upon the previous one, prompting students to apply modular software design to ensure the codebase easily extensible. (2) A creative virtual addressing scheme simplifies address resolution: device IDs (e.g., "R1") double as virtual MAC (vMac) addresses, while virtual IP addresses are encoded as subnetID.vMac (e.g., "net3.R1"). This formatting allows nodes to extract the target subnet and resolve the next-hop vMAC address via simple string parsing, eliminating the need for Address resolution Protocol or subnet masking. (3) Each node process (i.e., a virtual host, switch, or router instance) binds to a real host system's *IP:port* address, and links between adjacent nodes are simply emulated by neighbor's *IP:port* addresses.

The final deliverable is a fully functional internetwork: students launch multiple instances of their virtual host, switch, and router programs, potentially

across multiple laptops, to form a virtual network according to their topology file, and operate the virtual hosts to send and receive messages.

This project series has been successfully integrated into an undergraduate networking course over the past three years, receiving very positive student feedback and consistently being cited as a highlight of the course.

Teaching Practical Software Validation and Verification*

Work in Progress

Ramachandra B. Abhyankar
Indiana State University
Terre Haute, IN 47809
RB.Abhyankar@indstate.edu

The goal of software development is to deliver to the customer, software that meets the needs of the customer. The software developer is typically not familiar with the application domain of the customer's problem, and to learn about the domain and the problem, the developer often builds a "prototype." If the prototype gets the approval of the customer, the developer is reasonably sure that he is on the right track. The building of the prototype constitutes the "learning process" of the developer. There are several approaches to build and use the prototype. After the prototype has been built, the final product is built. This again can be done in many ways. The availability of tool-support to the developer to build the prototype and the final product can make a big difference. The involvement of the customer in the building of the prototype is crucial to ensure the success of the project. Tool support is becoming available to aid the software development process. It is the availability of tool support that makes validation and verification of the software practical. One approach to building a prototype is to use the VDM-SL language and the accompanying VDM toolbox. In Spring 2026 I taught a course in which model building using VDM-SL was taught to build a prototype. Also covered in the course was the use of the DAFNY tool to verify the correctness of code which is part of the final product. The talk will describe what I learned from my experience of using these tools. Also discussed will be possible alternative tools and methods for the software development task, to facilitate validation and verification.

Automatic Annotation of Prepositional Phrase Attachment in Globally Ambiguous Sentences*

Work in Progress

Ellis Cain[1] and Rachel Ryskin[2]
[1]Computing Program
Kenyon College
Gambier, OH 43022
cain1@kenyon.edu
[2]Cognitive and Information Sciences
University of California, Merced
Merced, CA 95343
rryskin@ucmerced.edu

Sentences can sometimes have multiple valid syntactic interpretations, leading to ambiguity. In "Pet the frog with the feather," the prepositional phrase could attach to the verb or to the noun. Verbs are biased to appear with particular syntactic constructions, which helps disambiguate meanings [2, 3]. The sentences used to study these biases usually have globally ambiguous with-PP attachment and require specialized hand-annotation, which is impractical for large-scale datasets. Here, we test methods for efficiently annotating sentence completions in the presence of global syntactic ambiguity.

We tested three automated methods: spaCy dependency parsing, BERT contextualized embeddings, and prompting with GPT 4o-mini. The spaCy method uses the PP's dependency head, and the BERT method uses the relative cosine distance between the PP and the potential attachment points (V/N; similar to [1]). We used two datasets of English sentence completions: Dataset 1 was collected through Amazon Mechanical Turk (1,805 completions; fully hand-annotated), and Dataset 2 through Prolific (79,791 completions). We used a subset of Dataset 1 as the training set (n = 1,600) and hand-annotated

a subset of Dataset 2 to serve as the test set (n = 2,360).

Accuracy was assessed using logistic regression with five-fold stratified cross-validation (averaging over 10 repetitions). The GPT method provided the most accurate annotations (Training: 92.3%; Test: 75.2%) compared to BERT (67.6%; 67.7%) and spaCy (51.6%; 52.2%). Interim results confirm that the traditional dependency parser failed in the presence of global ambiguity. While the BERT method improved performance by incorporating contextualized semantic information, the GPT method performed the best, likely due to the model scale and advanced reasoning capabilities. We are further testing whether LLMs' direct log-probabilities provide any additional benefit over prompting for annotation.

References

[1] Robert Hawkins et al. "Investigating Representations of Verb Bias in Neural Language Models". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Online: Association for Computational Linguistics, 2020, pp. 4653–4663. DOI: 10.18653/v1/2020.emnlp-main.376. URL: https://www.aclweb.org/anthology/2020.emnlp-main.376 (visited on 05/11/2026).

[2] Rachel A. Ryskin et al. "Verb Biases Are Shaped through Lifelong Learning." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 43.5 (5 May 2017), pp. 781–794. ISSN: 1939-1285, 0278-7393. DOI: 10.1037/xlm0000341. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/xlm0000341 (visited on 05/03/2026).

[3] Jesse Snedeker and John C. Trueswell. "The Developing Constraints on Parsing Decisions: The Role of Lexical-Biases and Referential Scenes in Child and Adult Sentence Processing". In: *Cognitive Psychology* 49.3 (Nov. 2004), pp. 238–299. ISSN: 00100285. DOI: 10.1016/j.cogpsych.2004.03.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0010028504000258 (visited on 05/05/2026).

Exploring Flexible Attendance Policies*

Work in Progress

David L. Largent
Department of Computer Science
Ball State University
Muncie, IN 47306
dllargent@bsu.edu

Over the years, I have incorporated many learner-centered aspects into my courses, such as allowing learners to help establish grading policies. For many years, I have gradually drifted towards making my approach to grading as equitable as possible, attempting to remove as many barriers to success as possible, and evaluating learners based on their demonstrated knowledge of the course material and not on their performance, such as meeting deadlines or attending class. Although we generally see a direct relationship between active participation in class and better grades, this relationship does not hold for all learners. Some learners succeed in class with less participation than others. Additionally, "life gets in the way" sometimes and causes learners to miss many class sessions, yet they can still demonstrate understanding of the course material. During the last few semesters, I have experimented with a variety of attendance policies that had little or no grade penalty attached to them. This ranged from having "tokens" the learner could "spend" to submit work late or miss an extra day of class, to having no direct impact on their grade. I will report briefly on my experiences and what changes I have made for this fall's courses.

FlowSpace: Closing the Loop Between Student Confusion and Faculty Action with AI*

Work in Progress

Sunho Kim, Grace Lee, and Alex Vo
Department of Computer Science
Denison University
Granville, OH 43023
{kim_s4, lee_g1, vo_a1}@denison.edu

When a student gets stuck on a programming concept, asking an AI assistant is now often faster than waiting for office hours or raising a hand in the next class. That convenience comes with a cost: the moment of confusion, which used to surface through questions, office-hour visits, or visible struggle in class, increasingly stays between the student and the chatbot. Professors lose the signal they once used to decide what to revisit, and students lose the chance to be routed to a person - an instructor or the academic resource center - who could resolve the underlying gap rather than just the immediate question. FlowSpace is built to restore that connection. After class, students submit a short note on what confused them. A clarifying step asks the student to specify the concept, step, or example involved, turning a vague note into something a professor can act on. Submissions are clustered by topic into a short digest delivered before the next class. The professor reviews it and decides what to do: revisit a concept, open with a targeted discussion question, or point a student toward office hours. AI's role is limited to clarifying and clustering; it does not answer the academic question, so every confused student is routed back to a person rather than another AI response. This work-in-progress reports on a Fall 2026 pilot with Denison University's Academic Resource Center (signed MOU), planned for four courses, including computer science sections, and approximately 80 students. The pilot tracks post-class submission rates, professor engagement with the digest, office-hour referral follow-through, and changes in

office-hour attendance. We are sharing this work with the CCSC:MW community to get feedback on the pilot's evaluation design from instructors who teach similar courses, and to explore whether this approach could extend to other liberal arts colleges in the region.

Integrating Computational Skill Development Through Student-Built Molecular Visualization Tools*

Work in Progress

Luiz Oliveira
Kenyon College
Gambier, OH 43022
oliveira3@kenyon.edu

This work-in-progress describes a curricular activity in which Computational Chemistry students design, build, and refine interactive, browser-based molecular visualization modules for use in General Chemistry courses. The project integrates two goals rarely combined at the undergraduate level of Chemistry, developing students' computational fluency through authentic software development practice and producing openly available educational tools grounded in rigorous scientific content.

Students will work in Jupyter notebooks using open-source Python libraries, including NGLview for three-dimensional molecular rendering, the Atomic Simulation Environment for atomic structure construction, RDKit for cheminformatics, and Plotly for interactive data visualization. Module topics are tied directly to the Computational Chemistry course, spanning thermodynamics, statistical mechanics, quantum mechanics, and spectroscopy. Students will write modular, reusable code intended for adaptation by other students, developing software design skills alongside a deeper understanding of chemical and mathematical formalism.

A central element is the feedback loop connecting developers to end users. Working prototypes will be piloted in General Chemistry lab sections, where Computational Chemistry students observe novice interactions, collect structured feedback, and revise their modules accordingly. This can also be interesting to students in Modern Physics and draws physics, chemistry, and

pre-health majors, bringing diverse perspectives on visualization design and audience-appropriate abstraction.

This project is in the conceptual and early planning stage, and implementation has not yet begun. Feedback is sought on several open questions: effective assessment strategies for measuring both developer learning and end-user comprehension, appropriate scope for individual student modules given small class sizes, and approaches for sustaining and maintaining student-built tools beyond the semester in which they are created.

Designing AI-Constrained Collaborative Activities for Computing Education Using coLearn-AI*

Conference Workshop

James Paul Skon[1,2]
[1]*Mathematics and Computer Science*
Kenyon College
Gambier, Ohio, USA
[2]*Karolinska Institutet*
Stockholm, Sweden
skonjp@kenyon.edu

Large Language Models (LLMs) make it possible for students to generate correct answers and working code while engaging minimally with the reasoning processes those outputs are intended to demonstrate. We describe this phenomenon as *cognitive bypass*. It presents a significant challenge for computing educators: although AI can support learning, unrestricted use can allow students to avoid the prediction, explanation, debugging, and reflection necessary for developing durable computing knowledge.

This three-hour workshop introduces coLearn-AI, a platform for designing and delivering collaborative computing activities in which AI supports student reasoning without replacing it. The platform structures collaboration through small-group work, turn-taking, shared responses, and constrained AI feedback. Rather than supplying solutions, the AI evaluates student reasoning and provides targeted prompts intended to help groups clarify, correct, or extend their answers before progressing.

Participants will first work in small groups to complete a collaborative activity within coLearn-AI. This experience will demonstrate the platform from the student perspective, including turn-taking, shared work, and AI-generated feedback. Participants will then use the platform's lightweight authoring for-

mat and AI-assisted authoring tools to design, implement, and test an activity appropriate for one of their own courses.

Workshop Outcomes

By the end of the workshop, participants will be able to:

- Explain how cognitive bypass can affect learning and assessment in computing education.
- Describe the principles of AI-constrained collaborative learning.
- Design and test a working coLearn-AI activity for use in a computing course.
- Plan the deployment of coLearn-AI in their own classrooms.
- Identify opportunities to participate in collaborative research and future publications involving the platform.

Workshop Schedule

The three-hour workshop will include the following components:

- **Introduction and System Overview (25 minutes).** Introduction to cognitive bypass, the design principles behind AI-constrained collaboration, and the coLearn-AI platform.
- **Hands-on Activity Experience (35 minutes).** Participants work in small groups to complete a structured collaborative activity using coLearn-AI.
- **Break (10 minutes).**
- **Activity Design and Implementation (60 minutes).** Participants use the platform's lightweight authoring format and AI-assisted authoring tools to create and test activities for their own courses.
- **Deployment and Classroom Integration (30 minutes).** Discussion of classroom deployment, group formation, progress monitoring, review of student responses, hosting options, and adaptation of existing instructional materials.
- **Research Collaboration and Discussion (20 minutes).** Discussion of lessons learned, participant questions, and opportunities to deploy coLearn-AI as part of ongoing research and future publications.

The intended audience is computing educators, particularly instructors of introductory programming courses, who want to incorporate AI into their teaching while preserving meaningful student engagement and reasoning. No previous experience with AI systems or activity-authoring tools is required.

Participants should bring a laptop with a modern web browser and WiFi access. No additional software installation is necessary. Workshop facilitators will provide access to coLearn-AI and all required materials.

AI-Assisted Instructional Material Creation *

Conference Tutorial

Charles Cusack
Department of Computer Science
Hope College
Holland, MI 49423
cusack@hope.edu

Recent advances in generative AI have made it possible for instructors to create textbooks, assignments, visualizations, assessments, programming exercises, and supplemental learning resources more efficiently. Used carefully, these tools can reduce development time, support open educational resources that lower student costs, and help instructors respond quickly to difficulties observed in class. Useful results, however, still require faculty to provide pedagogical direction, iteratively refine outputs, verify correctness, and remain responsible for the final content.

This tutorial presents practical, instructor-centered workflows for using generative AI to develop instructional materials, with particular emphasis on computer science education. AI tools are often especially effective for computing topics because algorithms, programming examples, pseudocode, documentation, mathematical explanations, and software discussions are widely represented in their training data. Drawing on the presenter's experience creating multiple open-source textbooks, interactive algorithm visualizations, assessments, and classroom activities, the session will include live demonstrations of end-to-end workflows used in actual course development. Examples will include generating and revising explanations, examples, homework and assessment problems, reading questions, active-learning activities, code samples, diagrams, interactive demonstrations, and targeted supplemental materials. The tutorial will also explore how AI can assist instructors in reviewing and analyzing student-submitted algorithms and code.

The session will emphasize iterative refinement, careful verification, and maintaining faculty control over content quality and pedagogical goals. It will also illustrate how AI can support broader faculty work, including writing, editing, organization, and communication tasks connected to teaching and course development.

The tutorial will demonstrate selected AI tools without endorsing any particular platform. Its focus is on transferable workflows and practices that can remain useful as individual tools and systems change. Participants will leave with concrete techniques, example workflows, and practical ideas they can incorporate into their own teaching while maintaining control over accuracy, pedagogy, authorship, and course goals.

Empowering Computer Science Education: A Hands-On Introduction to Tableau for Data Visualization *

Conference Tutorial

Mary Jo Geise, Helen Schneider
Department of Computer Science
University of Findlay
Findlay, OH 45840
geise@findlay.edu, schneider@findlay.edu

As the intersection of computer science and data analytics becomes increasingly vital in today's workforce, it's imperative to equip students with essential skills in data visualization. This workshop offers a hands-on introduction to Tableau, a leading business intelligence and analytics tool, designed to empower educators and students alike.

Through an interactive session and practical demonstrations, participants will gain foundational knowledge of Tableau's drag-and-drop functionality. They will learn to connect to the superstore sales data set, create bar chart, line chart and map visualizations, dynamic dashboards, and data narratives that effectively convey insights and tell the data story for complex datasets. The workshop will cover basic terminology and concepts, guiding participants through the process of transforming raw data into actionable insights.

Furthermore, the workshop will showcase how the University of Findlay has successfully integrated Tableau into its computer science curriculum. Examples of integration include the visualization of SQL queries, cluster analysis, forecasting statistical techniques, and the visualization ease of large datasets. Incorporating data visualization into traditional computer science education will better prepare students for the demands of the modern workforce. Participants are required to bring a laptop with Internet access. Detailed instructions on downloading a free trial version of Tableau will be provided to registered attendees prior to the workshop, ensuring readiness for the hands-on activities.

Biographies

Dr. Geise is professor emerita of computer science and director of the graduate program in applied security and analytics where she teaches data analytics classes. Dr. Schneider serves as associate professor and computer science department chair with interests in cybersecurity and outreach with academic and industrial partners.

The Transformative Impact of Artificial Intelligence in Teaching-Learning - The AI Effect in Academia *

Conference Tutorial

Bhaskar Sinha and Mohammad Amin
School of Technology & Engineering
National University
San Diego, CA 92123
{bsinha, mamin}@nu.edu

Abstract

Artificial Intelligence (AI) has become a defining force of the 21st century, and its applications extend far beyond business and industry. In academia, this research studies undergraduate Computer Science (CS) programs, where AI is revolutionizing the way we teach, learn, and evaluate, leading to transformative changes across all aspects of education. The integration of AI has also brought about changes in the ways students learn, and educators teach. With AI-driven tools and platforms becoming increasingly prevalent, it is essential to establish effective and accurate methods for evaluating learning in this "The AI Effect in Academia" environment. This research will explore the role of AI in CS, emphasizing its potential to enhance and evaluate learning, research, and administrative processes. It will further explore various strategies to assess the impact of AI on learning, ensuring that the benefits of AI technology are maximized while maintaining the integrity of educational evaluation.

Given that students now have access to AI tools, this effort is investigating new methodologies to measure student learning. The traditional methods are outdated, and the use of assignments, quizzes, discussions, and exams needs to be enhanced to evaluate student learning fairly and accurately. The areas

covered in this tutorial are: 1) Review currently used AI tools in education, 2) Personalized learning with AI, 3) AI effect on "How Learning Happens", and 4) Enhanced techniques for evaluating student-learning. Recent study with medical school students found no significant differences in student learning between synchronous and asynchronous online assessments[2]. But it was also observed that the average grades for online students are higher than both onsite and hybrid modalities. This raises challenging questions about how educators can assure the authentic learning of students in online modality while the students use AI tools during assessments. Julian Pakay reported that majority of the challenges faced while teaching an asynchronous biochemistry course were assessment issues[1].

References

[1] Julian Pakay. *The Challenge of Asynchronous Problem-Based Learning Online*. 2020. URL: https://www.linkedin.com/pulse/challenge-asynchronous-problem-based-learning-online-julian-pakay-/ (visited on 05/07/2026).

[2] Kuok Ho Daniel Tang. "Implications of artificial intelligence for teaching and learning". In: *Acta Pedagogia Asiana* 3.2 (2024), pp. 65–79.

Modules to Teach CS1 with Parallelism, Graphics, and a Network *

Conference Tutorial

David P. Bunde
Computer Science
Knox College
Galesburg, IL 61401
dbunde@knox.edu

Abstract

Modern computers are parallel, primarily run graphical applications, and make extensive use of online resources. That is the world our students know, but that reality is rarely reflected in CS1 assignments, which run serially and use only local text I/O. This tutorial will introduce participants to modules they can use to scaffold advanced features and make CS1 more closely reflect modern computing.

The first module is an unplugged activity, one that doesn't require a computer. Instead, students color "pixels" on paper to draw flags. The activity introduces parallel computing concepts such as speedup, dependencies, contention, system heterogeneity, and pipelining. It can also be tied to a programming activity on loops.

In the second module, students are given a library that provides "turtles" that draw trails in a 3D world. The students write code that generates JSON and uploads it. Their creations can be viewed with a modded Minecraft server or a web browser. From the student point of view, they write loops, but the discussion naturally includes basic networking and JSON. The library also supports parallelism, with multiple turtles able to draw faster than one.

The third module uses Greenfoot, an IDE designed for teaching Java in CS1. Even in a class primarily using a different IDE, Greenfoot is great for

introducing event-based programming. It is essentially a game framework with code attached to sprites, just like game design engines such as Unity or Godot except that it's aimed at CS1 students. The module introduces Greenfoot, provides sample code for common game features, and then has students create their own game. Students near the end of CS1 can make pretty sophisticated products.

Materials for these modules will be provided so that tutorial participants can adopt them.

Acknowledgements

This work was partially supported by the National Science Foundation through award OAC-2321020.

It Seemed Like a Good Idea at the Time: 2026 CCSC Midwest Edition*

Panel Discussion

James K. Huggins (moderator)[1], Paul Cerkez[2], Victoria Eisele[3], Benjamin Fine[4], Zachary Kurmas[5]

[1] *Kettering University, Flint, MI 48504*

jhuggins@kettering.edu

[2] *Coastal Carolina University, Conway, SC 29528*

pcerkez@coastal.edu

[3] *Front Range Community College, Fort Collins, CO 80526*

victoria.eisele@frontrange.edu

[4] *University of Wisconsin, Eau Claire, WI 54701*

finebt@uwec.edu

[5] *Grand Valley State University, Allendale, MI 49401*

kurmasz@gvsu.edu

1 Summary

We often learn more from failure than we do from success. However, the published computer science education literature has an understandable "success bias"; lessons learned from successful experiments are published, while lessons learned from failed experiments are not.

In this session, four panelists will share their experiences implementing ideas that "seemed like a good idea at the time", but ended with disastrous results. Each will share why their ideas seemed like a "good idea", the actual results, and the lessons learned from those failures.

2 Paul Cerkez

Student code often works well when tested on positive (expected) inputs, but not as well when tested on negative (unexpected) inputs. Our attempt to require negative testing for all final projects in software development courses seemed like a good idea, but encountered unexpected faculty resistance.

3 Victoria Eisele

We decided to offer our *Introduction to Programming* and *Computer Science I* courses as a combined, one-semester, 7-credit course. The combined course eliminated duplicated material between the two courses, and would allow students to move to advanced coursework (and complete their associate's degree) more quickly. Instead, the percentage of students who took advanced courses dropped from 70% to virtually none.

4 Benjamin Fine

We attempted to replace written final exams with one-on-one oral exams, giving students the opportunity to demonstrate their learning interactively and to learn from the assessment experience itself. Instead, what we experienced was student frustration, from either a broken appointment schedule or "tent cities" of students awaiting their examination.

5 Zachary Kurmas

We went "all-in" on alternative grading. It was mostly successful, with two major exceptions: (1) Using "meets expectations" and "not yet" instead of a percentage caused stress for many students (because they perpetually felt on the verge of failure), and (2) about 15% of our students struggled to meet expectations because they had always scraped by on partial credit.

6 Biographies

James K. Huggins is an Associate Professor at Kettering University.
Paul Cerkez is a Teaching Associate at Coastal Carolina University.
Victoria Eisele is an Instructor at Front Range Community College.
Benjamin Fine is an Associate Professor at the U. of Wisconsin–Eau Claire.
Zachary Kurmas is a Professor at Grand Valley State University.

Discussion on What AI Do We Need to Teach Future Computing Professionals*

Roundtable

Cathy Bareiss
Bethel University
Mishawaka, IN 46545
cbareiss@acm.org

This is a round table discussion as we all look to adapt our curriculum to deal with the changes AI has brought. Last year we discussed AI in general in the classroom. Now let's come together and figure out what AI skills our students need as computing professionals and how we scale our teaching of those skills. For example, when should our students ask AI to generate test cases?

Reviewers — 2026 CCSC Midwestern Conference

Imad Al Saeed ..... Saint Xavier University, Orland Park, IL
Stefan Brandle ..... Taylor University, Upland, IN
David P Bunde ..... Knox College, Galesburg, IL
Jennifer Jo Coy ..... Ball State University, Muncie, IN
David Furcy ..... University of Wisconsin Oshkosh, Oshkosh, WI
Michael C Gerten ..... Knox College, Normal, IL
Paul Gestwicki ..... Ball State University, Muncie, IN
Hector Hernandez ..... Benedictine University, Waukegan, IL
Deborah Hwang ..... University of Evansville (Retired), Evansville, IN
Zachary Kurmas ..... Grand Valley State University, GrandRapids, MI
David Largent ..... Ball State University, Muncie, IN
SugandhaMalviya ..... Ball State University, Muncie, IN
Jean Mehta ..... Saint Xavier University, Chicago, IL
Rangarajan Parthasarathy .....
..... University of Wisconsin Green Bay, GreenBay, WI
Ann Rangarajan ..... Illinois Institute of Technology, Chicago, IL
Takako Soma ..... Illinois College, Jacksonville, IL
James Vanderhyde ..... Saint Xavier University, Chicago, IL
Henry M. Walker ..... Grinnell College (Retired), Grinnell, IA
Fadi Wedyan ..... Lewis University, Romeoville, IL