The Journal of Computing Sciences in Colleges

Papers of the 32nd Annual CCSC Central Plains Conference

April 10th-11th, 2026
Drury University
Springfield, MO

Bin Peng, Associate Editor
Park University

Joan Gladbach, Regional Editor
University of Missouri Kansas City

Volume 41, Number 6 **April 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:

Bryan Dixon, President (2026), bcdixon@csuchico.edu, Computer Science Department, California State University Chico, Chico, CA 95929.

Shereen Khoja, Vice President/President-Elect (2026), shereen@pacificu.edu, Computer Science, Pacific University, Forest Grove, OR 97116.

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

Ed Lindoo, Treasurer (2026), 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.

Judy Mullins, Central Plains Representative (2026), mullinsj@umkc.edu, University of Missouri-Kansas City, Kansas City, MO (retired).

Michael Flinn, Eastern Representative (2026), mflinn@frostburg.edu, Department of Computer Science & Information Technologies, Frostburg State University, Frostburg, MD 21532.

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

David Largent, Midwest Representative (2026), 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 (2026), 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

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

Welcome to the 2026 CCSC Central Plains Conference

On behalf of the conference committee, it is our great pleasure to welcome everyone to our 32nd annual CCSC Central Plains regional conference in Springfield, Missouri hosted by Drury University. Each year, the conference brings together educators, researchers, and students who share a commitment to advancing computer science education, fostering scholarly exchange, and building a strong and supportive academic community across our region. We are delighted to continue this tradition in 2026 and are grateful for everyone's participation.

The conference serves as an important forum for the presentation and discussion of enduring challenges as well as innovative ideas in computer science education. This year's program reflects the breadth and vitality of our field, featuring peer-reviewed research papers, tutorials, workshops, panels, lightning talks, and nifty assignments on classroom techniques. The conference also highlights student research through student papers and poster sessions, providing valuable opportunities for mentoring, collaboration, and professional growth.

We are especially excited to introduce a keynote panel to this year's program (in lieu of a keynote speaker). The panel discussion will be on "Where We Going: The Future of Computing Education in the Face of AI", moderated by Dr. Chris Branton, Associate Professor of Computer Science at Drury University. This conversation brings educators and industry experts from across the region together to reflect on how rapidly evolving AI technologies are reshaping what we teach and the skills our students will need in the years ahead. We hope this panel sparks fresh ideas and collaborative approaches in computing education, and attendees will have the opportunity to direct questions to the panelists.

This conference would not be possible without the collective efforts of many people. We extend our gratitude to the authors who submitted their work and shared their ideas, the reviewers who ensured a high-quality program with a 68% paper acceptance rate, the committee members who are dedicated to planning and organizing the conference, and to the session moderators. The CCSC Board of Directors also deserves acknowledgment for their continued support of the conference. Without their financial support, the conference would not have been possible. Finally, we extend our gratitude to the administration and staff of Drury University and to the CCSC National Partners for their support.

We extend a very warm and delightful welcome to all presenters, attendees, and guests to the 2026 CCSC Central Plains Conference at Drury University. We encourage everyone to engage fully in the sessions, take advantage of opportunities to connect with colleagues and students, and enjoy the professional and personal interactions that make CCSC conferences so rewarding. We thank again all members of the Conference Committee who provided the necessary time and dedication to the conference with grace and commitment.

Padmavathi Iyer and Scott Sigman
Drury University
CCSC-2026 Central Plains Conference Co-Chairs

2026 CCSC Central Plains Conference Steering Committee

Conference Co-Chair
Padmavathi Iyer ..... Drury University, MO
Scott Sigman ..... Drury University (Retired), MO
Conference Secretary
Diana Linville ..... Northwest Missouri State University, MO
Pre-Conference Workshop Lead
Joan Gladbach ..... University of Missouri Kansas City, MO
Panels, Tutorials, Workshops Lead
Brian Hare ..... University of Missouri Kansas City, MO
Lightning Talks Lead
Wen-Jung Hsin ..... Park University, MO
Student Posters Lead
Joseph Kendall-Morwick ..... Washburn University, KS
Student Programming Contest Lead
Charles Riedesel ..... University of Nebraska-Lincoln (Retired), NE
K-12 Outreach Lead
Bill Siever ..... Washington University in St. Louis, MO
K-12 Outreach
Judy Mullins ..... University of Missouri–Kansas City (Retired), MO
Mahmoud Yousef ..... University of Central Missouri, MO
Aziz Fellah ..... Northwest Missouri State University, MO
Tiffany Ford ..... Ozarks Technical Community College, MO
Kevin Brunner ..... Graceland University, IA

Regional Board — 2026 CCSC Central Plains Region

Regional Rep
Judy Mullins ..... University of Missouri–Kansas City (Retired), MO
Regional Editor
Joan Gladbach ..... University of Missouri Kansas City, MO
Registrar
Ron McCleary ..... Retired
Regional Treasurer
Ajay Bandi ..... Northwest Missouri State University
Webmaster
Eric Manley ..... Drake University, IA
Secretary
Diana Linville ..... Northwest Missouri State University, MO
Publication Chair
Bill Siever ..... Washington University in St. Louis, MO

Reviewers — 2026 CCSC Central Plains Conference

Alrifai, Rad ..... Northeastern State University, Tahlequah, OK
Arrowsmith, Beth ..... University of Missouri - St. Louis, Saint Peters, MO
Attarwala, Abbas ..... California State University Chico, Chico, CA
Bourke, Chris Michael ..... University Of Nebraska-Lincoln, Lincoln, NE
Branton, Chris ..... Drury University, Springfield, MO
Brunner, Kevin ..... Graceland University, Lamoni, IA
Buerck, John ..... Saint Louis University, St. Louis, MO
Bunde, David ..... Knox College, Galesburg, IL
Carter, Karla ..... Bellevue University, Bellevue, NE
Chauhan, Kriti ..... Austin Peay State University, Clarksville, TN
Chiang, Chia-Chu ..... University Of Arkansas, Little Rock, AR
Copus, Belinda ..... University of Central Missouri, Warrensburg, MO
Feldhausen, Russell ..... Kansas State University, Shawnee, KS
Fellah, Aziz ..... Northwest Missouri State University, Maryville, MO
Ferguson, Ernest ..... Northwest Missouri State University, Maryville, MO
Furcy, David ..... University of Wisconsin Oshkosh, Oshkosh, WI
Hare, Brian ..... University of Missouri-Kansas City, Kansas City, MO
Herath, Suvineetha ..... Carl Sandburg College, Galesburg, IL
Hoot, Charles ..... Northwest Missouri State University, Maryville, MO
Horn, Trang ..... University Of Central Missouri, Warrensburg, MO
Hsin, Wen ..... Park University, Parkville, MO
Ifland, Jeff ..... State Of Nebraska, Lincoln, NE
Jones, James ..... Graceland University, Lamoni, IA
Kendall-Morwick, Joseph ..... Washburn University, Topeka, KS
Krishnaprasad, Srinivasarao .. Jacksonville State University, Jacksonville, AL
Li, Fang ..... Oklahoma Christian University, Edmond, OK
Linville, Diana ..... Northwest Missouri State University, Maryville, MO
Lu, Baochuan ..... Southwest Baptist University, Bolivar, MO
Manley, Eric ..... Drake University, Des Moines, IA
Metrolho, Jose .....
..... Polytechnic Institute of Castelo Branco, Castelo Branco, Portugal
Murella, Jhansi Priya ..... Saint Louis University, Saint Louis, MO
Pokorny, Kian ..... McKendree University, Lebanon, IL
Qin, Zhengrui ..... Northwest Missouri State University, Maryville, MO
Ravula, Swetha Reddy ..... Bucknell University, Lewisburg, PA
Riedesel, Charles ..... University of Nebraska - Lincoln, Beatrice, NE
Rogers, Michael ..... University of Wisconsin Oshkosh, Oshkosh, WI
Saquer, Jamil ..... Missouri State University, Springfield, MO

Siever, William ..... Washington University, St. Louis, MO
Tu, Cindy ..... Northwest Missouri State University, Maryville, MO
Urness, Timothy ..... Drake University, Des Moines, IA
Viotti, Paul ..... California State University Chico, Chico, CA
Walker, Henry ..... Grinnell College (retired), Napa, CA
Weber, Maria ..... Saint Louis University, St. Louis, MO
Wyne, Mudasser F ..... National University, San Diego, CA
Xing, Cong-Cong ..... Nicholls State University, Thibodaux, LA

Where Are We Going: The Future of Computing Education in the Face of AI?*

Keynote Panel Discussion

Chris Branton[1], Wen Hsin[2], Ying Cao[3],
Jason Barnes[4], Rahul Dubey[5]
[1]Math & Computer Science Department
Drury University, Springfield, MO 65802
cbranton@drury.edu
[2]Park University, Parkville, MO 64152
wen.hsin@park.edu
[3]School of Education and Child Development
Drury University, Springfield, MO 65802
ycao@drury.edu
[4]EDUCAUSE, Boulder CO 80301
jbarnes.drury.grad@outlook.com
[5]Computer Science Department
Missouri State University, Springfield, MO 65897
RaulDubey@MissouriState.edu

1 Summary

Computing is experiencing a moment of tremendous uncertainty. Is generative AI just the latest in a series of technological advances that periodically disrupt the practice of computing, or is it something more significant? Our panelists will engage the audience in a discussion of the impacts of AI on computing—and computing education—and how we can try to prepare for what is still to come.

2 Biographies

Dr. Chris Branton, Associate Professor, Drury University. (Panel Moderator)
Chris Branton is a faculty member in the Department of Mathematics and Computer Science at Drury University returning to academia after a career in custom software development and consulting. Dr. Branton's research interests include human computer interaction, software engineering, and game development technology. He is a long-time gamer and a serial hobbyist.

Dr. Wen Hsin, Professor, Park University
Dr. Wen-Jung Hsin is Professor of Computer Science at Park University for 22 years. Her teaching and research interest is primary in computer science education. She has published over 65 peer-reviewed articles. Given the rapid advancements in AI, she is deeply interested in and motivated by the question: Where will computing education go from here?

Dr. Ying Cao, Assistant Professor Education, Drury University
Dr. Cao is a faculty member in the School of Education and Child Development at Drury University. She holds a PhD in STEM Education and specializes in learning, pedagogy, and curriculum development in the STEM areas. Dr. Cao's work addresses cross-disciplinary topics such as cognition, attitudes, and simulation tools.

Mr. Jason Barnes, Data Analytics Engineer, EDUCAUSE Jason Barnes is an experienced software architect who helps maintain a Microsoft Fabric instance that integrates enterprise data for Power BI reporting and analytics. He has interests in games, home brewing, music, pets, and problem solving.

Dr. Rahul Dubey, Assistant Professor of Computer Science at Missouri State University
Dr. Rahul Dubey is a faculty member of Computer Science at Missouri State University. He holds a Ph.D. in Computer Science and Engineering from the University of Nevada Reno, and served as a postdoctoral research associate at the University of York in the United Kingdom. His research interests include explainable artificial intelligence, machine learning, and autonomous systems.

From EHRs to AI Agents. . . Healthcare Is Now a Technology First Industry*

Banquet Speech

Patrick Murfee
CoxHealth

Abstract

Patrick Murfee is Senior Vice President and Chief Information Officer at CoxHealth, where he leads enterprise information technology, digital strategy, cybersecurity, data and analytics, biomedical engineering, and clinical and business systems. With more than 25 years of experience across healthcare and technology, he is focused on advancing CoxHealth's adoption of automation, artificial intelligence, and emerging technologies through scalable, responsible innovation.

Prior to serving as CIO, Patrick was Chief Technology Officer at CoxHealth, helping modernize core infrastructure and prepare the organization for major enterprise initiatives, including large-scale ERP and EHR transformations. Before joining CoxHealth, he held senior leadership roles in both for-profit healthcare technology organizations and non-profit healthcare systems, leading multi-entity IT operations and complex enterprise programs. Patrick holds a Master of Health Administration from Ohio University and is an active member of the College of Healthcare Information Management Executives, contributing to state and national healthcare technology initiatives.

Mr. Murfee will share how his team is approaching technology transformation at CoxHealth, and why healthcare is quickly becoming a technology first industry. He will connect the evolution from EHR-centric work to automation, robotics, and AI agents, and how that shift is moving IT from a supporting

function to a core part of how care is delivered. Using real examples, he will translate these changes into the hybrid skills and mindset needed from the next generation of computer science graduates.

Utilizing Vibe Coding Techniques to Improve Prototyping in the Information Systems Analysis and Design Class*

Gary Yu Zhao and Cindy Zhiling Tu
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, MO 64468
{zhao, cindytu}@nwmissouri.edu

Abstract

Vibe coding, an emerging natural-language programming paradigm, lets developers specify functionality in plain English while AI generates code. This study examines how vibe-coding techniques enhance prototyping for Information Systems Analysis and Design (ISAD) students. Through structured Replit workshops, students practiced two prompting strategies (agile-style and waterfall-style) and built applications before applying the skills to a community-sponsored project. Using a qualitative case study that integrates workshop observations, cross-evaluations, and reflective discussions, we evaluate how vibe coding links analysis skills to prototype construction. Results indicate that vibe coding lowers technical barriers, shifts attention toward business needs and user experience, and improves client communication via iterative prototypes. Challenges include dependence on AI-generated code, quality assurance, and the need for stronger critical evaluation. The study offers guidance for IS educators modernizing pedagogy in AI-assisted development, positioning vibe coding as a practical means to effectively prepare students for agile project contexts while reinforcing analysis and design fundamentals.

1 Introduction

Information Systems Analysis and Design (ISAD) courses represent a critical component of information systems (IS) education, equipping students with the knowledge and skills necessary to translate business requirements into functional systems in the later Capstone projects [10]. While the core focus of ISAD is on business requirements analysis, system design principles, and client communication rather than technical programming, students still need to demonstrate their designs through working prototypes to effectively communicate with clients and gather critical feedback, which is a fundamental process in agile project management methodology.

Vibe coding, an emerging paradigm that leverages natural language prompts to create applications through artificial intelligence platforms such as Replit, Bolt, Windsurf, etc., represents a promising solution to this dilemma [9]. By enabling students to translate their design specifications and business logic into functioning prototypes without extensive programming experience, vibe coding allows ISAD students to focus their cognitive effort on analysis and design while maintaining the ability to deliver tangible prototypes that facilitate client feedback and iterative improvement. This research demonstrates how ISAD students utilized vibe coding techniques through structured workshops and a real-world project to enhance their prototyping capabilities. The study evaluates both the benefits and drawbacks of this approach, providing educators with evidence-based insights into the effective integration of AI-assisted coding within information systems education.

2 Methodology

2.1 Research Design and Theoretical Framework

This research is grounded in experiential learning theory, which emphasizes learning through concrete experience, reflective observation, abstract conceptualization, and active experimentation [6], [7]. The framework aligns naturally with the workshop-based pedagogy employed in this study, where students actively engage with the vibe coding tool (Replit), reflect on their experiences through discussion and evaluation activities, and apply learning to a complex real-world project. The study takes a design-based research approach, iteratively developing and refining workshop activities based on ongoing observations and student feedback [4].

2.2 Experiential Process and Implementation

Participants in this study were graduate and undergraduate students enrolled in an Information System Analysis and Design course at a Midwest university during the Fall 2025 semester. As shown in Figure 1, the research design incorporated three primary components: structured workshops introducing vibe coding techniques, a real-world project applying these techniques, and comprehensive evaluation through rubrics, client feedback, and peer assessment.

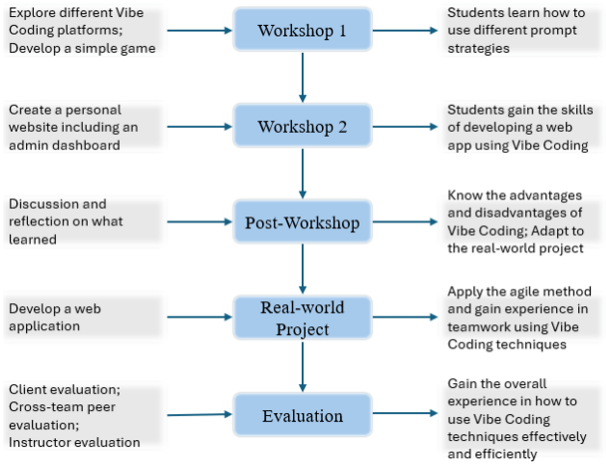

Figure 1: Process and Implementation

The first workshop introduced students to vibe coding concepts, explored several platforms, and practiced different prompting strategies through the development of a simple number-guessing game. Students were divided into groups and assigned to use either agile-style or waterfall-style prompting to create their games. Agile-style prompting involved providing Replit with general descriptions of desired functionality and then iteratively refining the game through multiple small prompts based on outputs received. Waterfall-style prompting required students to design the complete game functionality upfront and provide Replit with a comprehensive, structured description in a few detailed prompts. After developing their games, students participated in cross-evaluation activities where they played and assessed games created by peers using both prompting approaches.

The second workshop increased complexity by having students create personal websites with multiple interconnected pages and dynamic functional-

ity. Students designed and developed a website including an interactive admin dashboard. Evaluation criteria were expanded to include additional dimensions: aesthetic design quality, navigation effectiveness, content organization, responsiveness across devices, and proper implementation of authentication requirements. After completing both workshops, students participated in a facilitated discussion session designed to promote reflection and the synthesis of learning. The discussion addressed three main topics: (1) key lessons learned, (2) adaptation to a real-world project, and (3) advantages and disadvantages of vibe coding.

Following the workshops, students applied their developed vibe coding skills to a real-world project sponsored by the university's Facility Department. The project objective is to develop a role-based web application for managing recycling materials. Project teams must employ agile methods, including sprint planning, regular client demonstrations, iterative feedback collection, and prototypical refinement cycles.

Upon completion of the prototyping phase, multiple evaluation perspectives were gathered: - Client evaluation of each team's prototype against specified requirements. - Cross-team peer evaluation using standardized rubrics assessing functionality, usability, and adherence to requirements. - Instructor assessment of team collaboration, communication with the client, and iterative refinement.

Data analysis employed qualitative thematic analysis to identify patterns, themes, and insights across multiple data sources. The analysis process followed established procedures for qualitative research, beginning with data familiarization and progressing through systematic coding, theme development, and interpretation.

3 Results

The use of vibe coding for the ISAD course prototyping was well-received by students. The main findings demonstrate how the AI tools enhance students' learning outcomes and motivation.

3.1 Comparison of Prompting Strategies

As shown in Table 1, students using agile-style prompting required an average of 8 - 12 prompts to complete their number-guessing games, while those using waterfall-style prompting averaged 3 - 5 prompts. This difference reflects the fundamental nature of each approach: agile-style involves iterative

refinement through multiple small changes, while waterfall-style attempts comprehensive specification upfront. However, total development time showed less dramatic differences. Agile-style teams averaged 42 minutes to complete functional games, while waterfall-style teams averaged 38 minutes. The smaller time difference reflects the fact that waterfall-style teams invested substantially more time in their initial prompt design (averaging 15-18 minutes) before any code generation occurred, while agile-style teams began generating code within 3-5 minutes. Quality outcomes also showed nuanced differences. Waterfall-style prompting produced more complete initial implementations, with 78% of waterfall-prompted games meeting all specifications on the first generation compared to 31% of agile-prompted games. Student perceptions of the two approaches varied. In post-workshop reflections, 62% of students indicated a preference for agile-style prompting, citing greater flexibility and lower pressure. However, students who preferred waterfall-style (38%) emphasized the satisfaction of seeing more complete results immediately.

Table 1: Recycling Management System Project Client Evaluation Results

	Functional Coverage (Admin) 40 pts	Functional Coverage (Technician) 20 pts	Data & Reporting Accuracy 10 pts	Usability & Accessibility 10 pts	Authentication & Authorization 10pts	Performance & Reliability 10pts	Total 100 pts
Team 1	32	12	8	10	5	10	77
Team 2	30	20	8	8	5	10	81
Team 3	20	12	6	10	5	10	63
Team 4	36	20	10	10	10	10	96
Team 5	25	15	10	8	8	10	76
Team 6	38	20	10	10	8	10	96
Team 7	30	20	10	10	8	10	88

Interestingly, in Workshop 2 and the Recycling project, when students could choose their approach, most (71%) adopted hybrid strategies combining elements of both.

3.2 Effectiveness of Vibe Coding for Prototyping

For the Recycling Management System project, the evaluation criteria are shown in Figure 2. Satisfying the functional requirements was prioritized, i.e., 40% and 20% for the admin and technician roles, respectively. Other criteria, including data & reporting accuracy, usability & accessibility, authentication and authorization, and performance & reliability, each take 10% evenly.

As shown in Table 2, all seven project teams successfully delivered func-

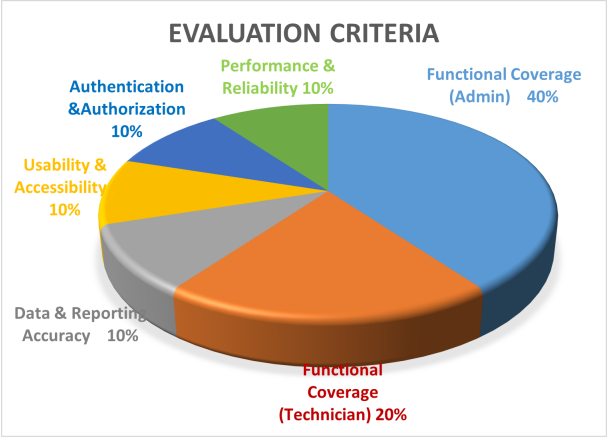

Figure 2: The Evaluation Criteria for the Real-world Project

tional prototypes implementing the core requirements. Average functional completeness scores from client evaluation were 82/100, with two teams achieving complete implementation of all specified features and successfully implementing 96% of requirements. The client reported high satisfaction with the prototyping process and final deliverables. The client particularly appreciated the rapid development cycle facilitated by vibe coding, which enabled quick iteration through multiple design possibilities and responsive incorporation of feedback.

Table 2: The Comparison of Prompting Strategies in Workshop 1

	Number of Prompts (times)	Development time (minutes)	Initial Prompts time (minutes)	Initial Prompt Requirement Coverage	Project Team Preference
Waterfall-style Prompting	3-5	38	15-18	78%	38%
Agile-style Prompting	8-12	42	3-5	31%	62%

3.3 Benefits and Challenges of Vibe Coding in Prototyping

Multiple benefits and challenges of vibe coding for ISAD prototyping emerged from analysis of workshop/project outcomes and student reflections. The ben-

efits include:

Reduced Technical Barriers: Students who previously struggled with programming syntax and debugging could create working prototypes, enabling participation in complete design-to-implementation cycles.

Accelerated Prototyping: This acceleration enabled more iterations within limited class time, supporting agile development principles of rapid feedback and continuous improvement.

Enhanced Focus on Design and Requirements: By automating code generation, students could concentrate their cognitive efforts on business analysis, user experience design, and client communication rather than debugging syntax errors.

Improved Communication Skills: The requirement to describe desired functionality clearly through natural language prompts developed students' requirements specification abilities. Students learned to think.

The challenges of using vibe coding techniques in the ISAD class include:

Over-Reliance on AI: Some students developed excessive dependence on AI-generated code without developing understanding of underlying logic. When AI outputs contained errors or did not match intentions, these students struggled to diagnose problems.

Debugging Challenges: When AI-generated code did not work correctly, students often struggled to identify and correct problems. Some students responded to bugs by simply trying different prompts randomly rather than systematically diagnosing issues.

Quality Control Concerns: AI-generated code sometimes included unnecessary complexity, inefficient algorithms, or subtle bugs that students lacked the expertise to identify. Without strong programming backgrounds, students could not effectively evaluate code quality beyond surface-level functional testing.

Limited Learning of Fundamentals: Some faculty observers expressed concern that vibe coding might allow students to skip learning programming fundamentals that provide important context for systems design decisions.

4 Discussion

4.1 Pedagogical Impact and Implications

The results substantiate research findings that AI-assisted development can improve learning outcomes when properly structured around knowledge construction and augmentation rather than procedural application [2]. However, the effectiveness of vibe coding as a learning tool depends significantly on how it is positioned within the broader curriculum. When vibe coding is presented as a supplement to, rather than replacement for, programming fundamentals, it can accelerate learning by allowing students to create sophisticated prototypes while still developing an understanding of underlying technical concepts [6], [2].

The client's positive assessment of the prototyping process and the rapid iteration capability merit particular attention. In real-world systems development, the ability to quickly incorporate client feedback and adapt designs represents a critical success factor. The accelerated iteration cycle that vibe coding enabled directly contributes to this practical advantage. The study's findings align with experiential learning theory's emphasis on concrete experience, reflection, conceptualization, and experimentation [6], [7]. The structured workshop progression supported movement through this learning cycle multiple times, reinforcing key concepts.

4.2 Prompting Strategies and Development Methodologies

The comparison between agile-style and waterfall-style prompting illuminates important connections between vibe coding and broader software development methodologies. Students' gravitation toward hybrid approaches suggests that strict adherence to either extreme may be less effective than flexible adaptation to specific contexts [8]. The finding that waterfall-style prompting produces more complete initial implementations but requires longer upfront planning reflects fundamental tradeoffs between the methodologies. From a pedagogical perspective, both approaches offer value. The finding that both approaches produce comparable quality outcomes suggests that students might benefit from exposure to and practice with both methodologies, developing flexibility in their approach to different project contexts.

4.3 Skill Development and Potential Risks

While the results demonstrate significant benefits of vibe coding integration, the study also identifies important considerations regarding skill development

and long-term learning outcomes. Concerns about potential skill decay and hindered skill development identified in research on AI assistance represent legitimate pedagogical considerations requiring careful attention [1]. This issue parallels broader debates in computing education about how much students need to understand implementation details versus higher-level abstractions [5], [3]. For ISAD students progressing to more technical computing courses or roles, explicit attention to foundational programming concepts may be advisable. Instructors might consider supplementing vibe coding-based prototyping with selective exposure to underlying code to ensure students develop an understanding of implementation approaches and underlying logic, even if they don't hand-code complete applications.

5 Conclusion

Vibe coding streamlines prototyping in ISAD by lowering technical barriers, so students can focus on analysis, design, and client communication. Structured workshops build prompting skills; agile and waterfall prompting each offer distinct pedagogical value. In a recycling management project, students produced functional prototypes that met client needs within constraints, achieving rapid iteration. The approach broadens participation for less technical students but still requires grounding in programming and architecture. Effective courses should scaffold prompting, use assessments, align with agile, balance AI with coding, and train critical evaluation of AI outputs. Future research should study learning outcomes and compare AI-assisted vs traditional ISAD education.

References

[1] B. N. Macnamara et al. "Does using artificial intelligence assistance accelerate skill decay and hinder skill development without performers' awareness?" In: *Cognitive Research: Principles and Implications* 9.1 (2024), p. 46. DOI: 10.1186/s41235-024-00572-8.

[2] J. Prather et al. "The Robots are Here: Navigating the Generative AI Revolution in Computing Education". In: (2023), pp. 108–159. DOI: 10.1145/3623762.3633499.

[3] M. Kazemitabaar et al. "Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming". In: Apr. 2023, pp. 1–23. DOI: 10.1145/3544548.3580919.

[4] S. Barab and K. Squire. *Design-Based Research: Putting a Stake in the Ground*. Psychology Press, 2016, pp. 1–14.

[5] B. A. Becker et al. "Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation". In: vol. 1. 2023, pp. 500–506. DOI: 10.1145/3545945.3569759.

[6] D. A. Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs, N.J: Prentice Hall, 1984.

[7] M. G. Morris, V. Venkatesh, and P. L. Ackerman. "Gender and Age Differences in Employee Decisions about New Technology: An Extension to the Theory of Planned Behavior". In: *IEEE transactions on engineering management* 52.1 (2005), pp. 69–84.

[8] J. C. Pereira and R. de F. S. M. Russo. "Design Thinking Integrated in Agile Software Development: A Systematic Literature Review". In: *Procedia computer science* 138 (2018), pp. 775–782. DOI: 10.1016/j.procs.2018.10.101.

[9] P. P. Ray. "A Review on Vibe Coding: Fundamentals, State-of-the-art, Challenges and Future Directions". In: *TechRxiv* (May 2025). URL: https://www.techrxiv.org/doi/full/10.36227/techrxiv.174681482.27435614.

[10] G. Y. Zhao and C. Z. Tu. "Adapt DevOps Method to Information Systems Capstone Course Projects". In: *Journal of Computing Sciences in Colleges* 40.6 (2025), pp. 18–28.

Teaching Multiple Paradigms for Intelligent Systems: Integrating Search, Knowledge-Based Reasoning, and Machine Learning in Graduate Education*

Fang Li
Computer Science Department
Oklahoma Christian University
Edmond, 73013
fang.li@oc.edu

Abstract

This paper presents a pedagogical approach for teaching intelligent systems that systematically integrates three fundamental paradigms: search-based agents, knowledge-based agents, and machine learning-based agents. We describe a graduate-level course designed for intensive seven-week delivery where students develop three major group projects concurrently while completing theoretical homework assignments. A distinctive feature is substantial coverage of declarative programming (Prolog and Answer Set Programming), which students report as uniquely valuable and unavailable in typical computer science programs. The course also integrates responsible use of AI-assisted programming tools, requiring comprehensive documentation of LLM usage while maintaining accountability through examinations and demonstrations. Our findings demonstrate that this multi-paradigm approach develops systematic understanding of intelligent systems and equips students to select appropriate computational approaches based on problem characteristics.

1 Introduction

The field of artificial intelligence has experienced remarkable growth, with machine learning dominating both research and educational curricula. While these advances have produced impressive capabilities, they have created an imbalance in AI education. Many contemporary AI courses focus almost exclusively on statistical learning methods, providing limited exposure to alternative approaches that remain valuable for specific problem classes.

This pedagogical shift has created students who view AI primarily through the lens of machine learning, often unaware of other computational paradigms. Students may graduate without understanding that many problems are more effectively solved through systematic search, logical reasoning, or hybrid approaches. This narrow perspective limits problem-solving flexibility and leaves gaps in understanding AI's conceptual foundations.

The challenge becomes acute in graduate programs where students arrive with diverse backgrounds and objectives. Furthermore, the proliferation of AI-assisted programming tools, particularly Large Language Models (LLMs) capable of generating code, presents both opportunities and challenges. Educational approaches must teach responsible use of AI tools while maintaining emphasis on conceptual understanding.

We developed a graduate course titled "Intelligent Systems" that integrates three fundamental paradigms: search-based problem solving, knowledge-based reasoning, and machine learning. Students complete theoretical homework assignments reinforcing core concepts while working in teams to develop three major projects concurrently: a PACMAN game using search algorithms, knowledge-based systems in Prolog and ASP, and an ensemble learning application. All projects are presented in a culminating demonstration session during the final week.

A distinctive feature is substantial coverage of declarative programming and logic-based reasoning. Student feedback consistently identifies this content as uniquely valuable, with many reporting these approaches are unavailable in other courses. The course explicitly addresses AI-assisted programming by permitting LLM use while requiring comprehensive documentation, recognizing that students will use these tools professionally while teaching critical evaluation skills.

2 Related Work

Traditional AI education follows frameworks like Russell and Norvig [8], covering search, knowledge representation, reasoning, and introductory machine learning. Specialized machine learning curricula emphasize statistical founda-

tions [7] with contemporary deep learning extensions [4]. Recent courses target specific areas like deep learning [9] and natural language processing [6].

Logic programming has its own tradition, with courses teaching Prolog [2] and Answer Set Programming [3], but these are often taught separately from AI curricula. Recent work examines teaching rapidly evolving AI technologies [5] and integration of AI code generation tools [1].

Our work extends these foundations by proposing a structured curriculum that explicitly connects search-based, knowledge-based, and learning-based approaches within a coherent framework while integrating AI-assisted programming tools with appropriate accountability measures.

3 Course Design and Architecture

3.1 Multi-Paradigm Structure

The course employs four major components:

Part 1: Introduction to AI establishes foundational concepts and the agent paradigm as a unifying framework. Students explore definitions of AI, historical context, and the three computational approaches structuring subsequent learning.

Part 2: Search-Based Agents develops competencies in state-space search including uninformed strategies (breadth-first, depth-first, iterative deepening), informed search using heuristics (greedy best-first, A*), local search methods (hill climbing, simulated annealing, genetic algorithms), and adversarial search (minimax, alpha-beta pruning).

Part 3: Knowledge-Based Agents introduces declarative programming through two modules. Monotonic reasoning with Prolog teaches logic programming fundamentals: facts, rules, queries, unification, backtracking, and recursive problem solving. Non-monotonic reasoning with Answer Set Programming extends logical reasoning to handle incomplete information, default assumptions, and reasoning with exceptions using the Clingo system.

Part 4: Machine Learning-Based Agents covers classical machine learning (decision trees, support vector machines, neural networks) and deep learning (convolutional and recurrent networks), with emphasis on ensemble learning methods combining multiple models for improved performance.

This structure enables students to appreciate how different approaches address different problem characteristics. Search excels when state spaces can be enumerated; logic-based reasoning handles domains with complex rules; machine learning extracts patterns from data.

3.2 Pedagogical Framework

Our approach is guided by core principles:

Parallel Structure creates consistency across paradigms. Each follows identical pedagogical sequence: lectures introducing theoretical foundations, homework assignments reinforcing concepts, and collaborative group projects applying the approach. This parallel structure enables direct comparison of paradigms' strengths and limitations.

Theory-Practice Integration combines rigorous conceptual explanations with hands-on implementation. Students complete theoretical homework deepening understanding while simultaneously implementing concepts through group projects.

Authentic Problem Contexts ground projects in realistic scenarios: search projects involve game-playing agents; knowledge-based projects implement expert systems or planning applications; machine learning projects analyze real datasets.

Comparative Analysis encourages evaluating trade-offs between approaches. Students develop judgment about paradigm selection for specific problem types.

Responsible AI Tool Integration recognizes LLMs have become ubiquitous in software development. Rather than prohibiting use, we teach responsible usage while maintaining accountability for understanding through documentation requirements and examinations.

3.3 Intensive Seven-Week Format

The course delivers 3.5-hour weekly sessions over seven weeks (24.5 contact hours). This compressed timeline creates urgency maintaining student engagement while enabling connections across paradigms. Students work on all three projects concurrently, applying concepts as introduced. We address timing constraints by focusing on fundamental concepts with enduring value, utilizing flipped classroom elements, and providing extensive asynchronous resources.

4 Assessment Strategy

The course employs multi-faceted assessment:

Homework Assignments (20%) consist of theoretical problems for each major topic: algorithm analysis, complexity calculations, problem formulation exercises, logic programming traces, and machine learning concept problems. These ensure individual accountability and concept mastery.

Group Projects (50%) engage students in three collaborative endeavors developed concurrently throughout the semester. Students work in consistent

teams of 3-4 members:

Project 1: Search-Based PACMAN Agent requires developing an automated PACMAN game where the character navigates independently using search algorithms (breadth-first, depth-first, A*, or uniform cost search). Students use Python and Pygame, implementing AI agent design, environment analysis through maze representation, real-time pathfinding, and obstacle avoidance.

Project 2: Knowledge-Based Systems consists of two subprojects. Part 1 develops a Prolog program demonstrating logical reasoning (expert systems, path finding, puzzle solvers, or planning). Part 2 creates an ASP solution using Clingo for constraint satisfaction, planning, or scheduling problems. Students may solve the same problem in both paradigms to enable direct comparison.

Project 3: Machine Learning with Ensemble Methods requires developing a solution using ensemble learning on real-world datasets from Kaggle. Students implement at least two ensemble methods (random forest, gradient boosting, bagging, or voting), performing data preprocessing, model development, and performance comparison.

The final week features a demonstration session where each team presents all three projects together, enabling comprehensive assessment, comparative understanding, peer learning, and professional development.

Midterm Examination (15%) assesses theoretical understanding of search and knowledge-based approaches through problem formulation, algorithm analysis, logic programming concepts, and comparative questions. Students complete exams individually without LLM access.

Final Examination (15%) evaluates comprehensive understanding across all paradigms, emphasizing comparative analysis and integrated understanding.

All projects require GitHub for version control and comprehensive documentation including README files, inline comments, architecture documentation, LLM usage documentation, and evaluation reports.

5 AI-Assisted Programming Integration

A distinctive feature is explicit integration of AI-assisted programming tools. Rather than prohibiting LLMs, we embrace them while requiring transparency and maintaining accountability.

5.1 Pedagogical Rationale

Reflecting Professional Practice: AI-assisted tools like GitHub Copilot and ChatGPT are ubiquitous in industry. Preparing students requires teaching effective use rather than pretending they don't exist.

Lowering Implementation Barriers: LLMs help students overcome syntax hurdles, particularly valuable for unfamiliar languages like Prolog and ASP, allowing focus on conceptual understanding.

Teaching Critical Evaluation: Requiring students to document and validate LLM-generated code teaches critical evaluation skills essential for working with any external code source.

Enhancing Learning Through Iteration: LLMs enable rapid experimentation with different approaches, often leading to deeper understanding through comparison.

5.2 Documentation Requirements

Students must comprehensively document LLM usage:

- Identify which LLMs were used for each project component
- Specify exactly which code parts received LLM assistance
- Include prompts provided to LLMs and responses received
- Explain validation performed, testing conducted, and modifications made
- Demonstrate understanding through explanations written in own words

These requirements create accountability for understanding, develop professional practices, provide assessment transparency, and teach prompt engineering skills.

5.3 Assessment Implications

Projects emphasize higher-level skills like problem formulation, architecture design, component integration, and result evaluation—skills requiring understanding beyond code generation. The demonstration session verifies understanding through live interaction and questioning. Examinations assess conceptual knowledge without LLM access, ensuring AI-assisted project work corresponds to genuine understanding. Documentation quality becomes part of project assessment.

6 Learning Outcomes and Evaluation

6.1 Student Performance and Feedback

Implementation across multiple offerings has demonstrated encouraging outcomes:

Systematic Understanding: Students report gaining comprehensive perspective on building intelligent systems, understanding multiple valid approaches exist and paradigm selection should match problem characteristics. Representative comments include: "I now understand that AI isn't just machine learning—there are completely different ways to create intelligent behavior."

Unique Value of Declarative Programming: Students consistently report that Prolog and ASP exposure was uniquely valuable: "I had never seen anything like Prolog before—it completely changed how I think about programming" and "ASP made me realize that not every problem needs machine learning; sometimes logical rules are clearer and more maintainable."

Paradigm Comparison Ability: The parallel structure helps students develop judgment about approach selection. Teams explicitly justify paradigm choices during demonstrations, showing sophisticated analysis of problem-approach matching.

Problem Formulation Appreciation: Students develop understanding that effective problem formulation often matters more than algorithm sophistication across all paradigms.

Positive LLM Integration Experience: Students report the documentation requirements struck appropriate balance: "Being able to use ChatGPT for syntax help let me focus on understanding concepts rather than fighting with unfamiliar languages" while "The documentation requirements made sure I understood the code even when I got help generating it."

6.2 Evidence of Learning

Project Quality: Final projects demonstrate sophisticated application showing genuine understanding. Quality has remained consistent despite LLM availability, suggesting documentation requirements and demonstrations maintain accountability.

Examination Performance: Students perform well on both exams, particularly on comparative analysis questions. Examinations successfully verify that potentially LLM-assisted project work corresponds to genuine conceptual understanding.

Documentation Quality: LLM usage documentation reveals significant engagement with generated code, showing iteration, validation, modification, and understanding rather than blind acceptance.

Demonstration Performance: Teams effectively present work, answer implementation and design questions, and explain both what systems do and why particular choices were made.

6.3 Implementation Challenges

Heterogeneous Backgrounds: Students arrive with varying programming experience, mathematical preparation, and AI exposure. We address this through supplementary resources, peer mentoring, and flexible scaffolding. LLM availability helps students with weaker programming skills overcome syntax barriers.

Paradigm Shift Difficulty: Logic programming creates cognitive load for students accustomed to imperative programming. We provide extensive examples, interactive demonstrations, and explicit comparisons. LLMs enable personalized support through real-time explanations.

Time Constraints: The seven-week timeline requires careful content curation, prioritizing fundamental concepts with enduring value.

Group Dynamics: We implement project management requirements, interim checkpoints, and peer evaluation to promote equitable collaboration. GitHub histories provide contribution visibility.

Maintaining Coherence: Concurrent project development, consistent parallel structure, and the integrated demonstration session help students see connections rather than disconnected units.

7 Best Practices and Insights

Several key insights emerged through multiple iterations:

Parallel Structure Facilitates Learning: Consistent pedagogical sequence helps students develop meta-learning skills and facilitates comparison.

Declarative Programming Deserves Curriculum Space: Despite emphasis on machine learning, logic programming retains substantial value. Given diminishing coverage in modern curricula, deliberate inclusion proves important.

Comparison Develops Judgment: Teaching multiple approaches and requiring comparison develops judgment more valuable than deep expertise in any single approach.

Concurrent Projects Enhance Integration: Working on all projects simultaneously rather than sequentially helps maintain awareness of all paradigms and see connections clearly.

Demo Sessions Enhance Learning: Presenting all projects together exposes students to diverse strategies, creates motivation for quality, and reinforces learning about paradigm selection.

Embrace AI Tools with Accountability: Permitting LLM use with documentation requirements and rigorous assessment represents sustainable approach to AI-assisted programming, preparing students for professional practice while ensuring genuine learning.

Theoretical Grounding Remains Essential: Despite LLM availability, theoretical homework and examinations remain crucial for conceptual understanding.

8 Future Directions

As AI continues evolving, several areas merit enhancement:

Hybrid Approaches: Greater emphasis on systems integrating multiple paradigms could prove valuable, aligning with contemporary neuro-symbolic research.

Expanded LLM Integration: Exploring advanced uses beyond code generation—as interactive tutors, brainstorming partners, or code review assistants—could enhance learning.

Enhanced Industry Connections: Partnerships providing guest lectures, project mentorship, and real-world scenarios would strengthen understanding of professional deployment.

Assessment Innovation: Reflective writing, mini-research projects, or case studies could enhance learning.

Research on LLM Impact: Systematic research comparing cohorts before and after LLM integration would contribute to pedagogical literature.

9 Conclusion

Our multi-paradigm course addresses gaps in AI education by systematically integrating search-based, knowledge-based, and learning-based approaches while reflecting contemporary software development practices including responsible AI tool use.

Key contributions include: (1) structured progression building explicit connections between three fundamental AI paradigms, (2) substantial coverage of declarative programming students identify as uniquely valuable and unavailable elsewhere, (3) parallel pedagogical structure facilitating comparison, (4) integration of theoretical homework with three major concurrent collaborative projects culminating in comprehensive demonstration, (5) responsible integration of AI-assisted programming through documentation requirements and maintained accountability, and (6) demonstration that intensive seven-week format can effectively deliver multi-paradigm content.

Students develop systematic understanding transcending familiarity with individual techniques, gaining ability to analyze problems, identify appropriate approaches, and implement solutions using diverse methodologies. Student feedback emphasizes unique value of knowledge-based content and appreciation for realistic LLM integration with appropriate accountability.

The course demonstrates that balanced coverage of multiple AI paradigms within compressed timelines is possible without sacrificing depth or practical skill development. The parallel structure with coordinated assessments creates experiences developing both technical competency and professional judgment. LLM integration with accountability offers a model for computing education more broadly.

As AI reshapes industries and society, educational approaches equipping students with diverse problem-solving tools become increasingly valuable. Students prepared to select appropriate approaches based on problem characteristics rather than defaulting to familiar methodologies demonstrate greater adaptability. Our experience suggests multi-paradigm approaches can prepare students for dynamic careers while providing conceptual foundations remaining relevant despite rapid technological change.

Future work will focus on enhancing paradigm integration, exploring hybrid approaches, strengthening industry connections, investigating scalable delivery methods, and conducting systematic research on LLM integration's impact on learning outcomes.

References

[1] Brett A Becker et al. "Programming Is Hard–Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education* (2023), pp. 500–506.

[2] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. 4th. Harlow, UK: Addison-Wesley, 2012.

[3] Martin Gebser et al. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[5] Amanda M Holland-Minkley, Thomas E Lombardi, and Sharon M Carver. "Teaching AI in a Liberal Arts Context: Challenges and Opportunities". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*. ACM. 2023, pp. 312–318.

[6] Dan Jurafsky and James H Martin. *Speech and Language Processing*. 3rd. Draft available at https://web.stanford.edu/ jurafsky/slp3/. Upper Saddle River, NJ: Prentice Hall, 2023.

[7] Tom M Mitchell. *Machine Learning*. New York, NY: McGraw-Hill, 1997.

[8] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th. Hoboken, NJ: Pearson, 2020.

[9] Aston Zhang et al. *Dive into Deep Learning*. Cambridge, UK: Cambridge University Press, 2023.

A Literature Review of AI Applications in Higher Education*

Trang Horn and Mahmoud Yousef
Computer Science and Cybersecurity
University of Central Missouri
Warrensburg, MO 64093
{thorm,yousef}@ucmo.edu

Abstract

Artificial Intelligence (AI) has a significant impact on most industries, and education is not an exception. This systematic literature review explores the application of AI in higher education, with a focus on identifying its benefits, challenges, and recommendations for integrating AI in the classroom. The study utilizes the Education Resource Information Center (ERIC), a digital library maintained by the U.S. Department of Education, as well as Google Scholar and Semantic Scholar. Using the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines, 24 peer-reviewed articles from three databases were analyzed following a rigorous screening process. Findings indicate that AI can be a valuable tool for both students and faculty. AI can assist academic research for both faculty and students, providing intelligent tutoring systems, enabling adaptive learning, improving student engagement and performance, and providing quick real-time feedback to students via a chatbot. AI can also assist faculty with automating repetitive tasks such as developing syllabi, generating test questions, and evaluating students' work. Despite these advantages, several challenges arise when leveraging AI in higher education. The challenges include resistance to change, lack of institutional leadership support, maintaining academic integrity, ensuring consistency and fairness in AI-powered assessment tools, data privacy issues, and the lack of institutional policies

and guidelines related to AI use. To address these challenges, institutions are recommended to increase funding, build AI expertise, and enhance their technological infrastructure to implement and leverage AI integration effectively. In addition, higher education faculty should consider reforming curriculum and assessment practices to adapt to the new adoption of AI in higher education. Furthermore, it is crucial to establish clear institutional policies for the acceptable and ethical use of AI. Finally, the paper proposes a sequence of steps that higher institutions can follow to implement AI effectively.

1 Introduction

Artificial Intelligence (AI) is a technology that allows machines to simulate human learning, problem-solving, and decision-making, allowing for both creativity and autonomy [20]. AI has had significant impacts across multiple sectors, including higher education. The field of AI was established in the 1950s when Alan Turing published his paper called *Computing Machinery and Intelligence*, which introduced a new concept that is now known as the Turing Test or a measure of machine intelligence. The term "artificial intelligence" was formed during this period [21].

Despite growing adoption, the application of AI in higher education remains fragmented. Existing studies about the adoption of AI in higher education vary in focus, making it difficult for educators to gain a comprehensive understanding of how AI is being used in higher education, what challenges it introduces, and what recommendations for institutions that start to adopt AI.

This paper seeks to address this gap by conducting a systematic literature review of AI applications in higher education and answering the following research questions:

1. What are the applications of AI in higher education?
2. What challenges does AI adoption face in higher education?
3. What are the recommendations for higher education institutions regarding the adoption of AI?

2 Methodology

In this literature review, we conduct a systematic literature review of research papers on the possible application of AI in higher education. To ensure transparency and reproducibility, we follow the *Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA)* process. PRISMA involves

four key stages: identification (search record), screening (abstract-level review), eligibility (full-text assessment), and inclusion (final selection of studies) [13].

At each stage, articles were carefully examined for relevance and methodological quality. The PRISMA flow diagram in Figure 1 illustrates the number of articles identified, screened, excluded, and ultimately included in the final review.

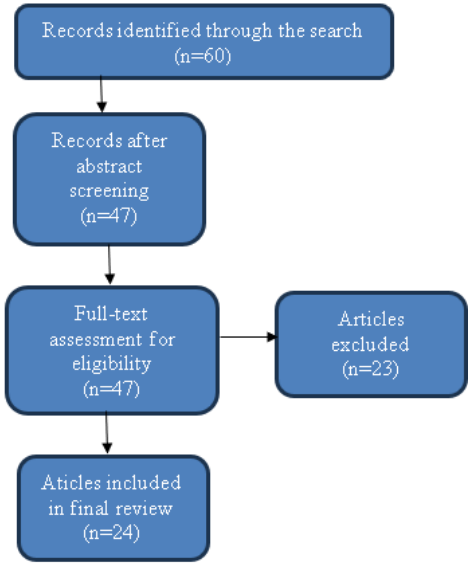

Figure 1: PRISMA flow diagram

This literature review was conducted using the *Education Resources Information Center (ERIC)* database. ERIC is a widely recognized digital database for peer-reviewed educational research. In addition, we also use Google Scholar and Semantic Scholar. Google Scholar and Semantic Scholar provide a wide coverage of multidisciplinary research in computer science, artificial intelligence, and higher education, which was important for our study. To form the search query, Boolean operators were applied to ensure a comprehensive retrieval of relevant articles published in 2025 and 2024. In ERIC, we used the query ("artificial intelligence" OR "AI") AND ("higher education" OR "college" OR "university"), limiting the results to peer-reviewed, full-text articles from 2024 to 2025. The same keyword combination was applied in Google Scholar and Semantic Scholar, with the year filter set from 2024 to 2025. This time frame was chosen to ensure the reviewed studies reflected the most recent de-

velopments. From the search results, the top 60 most relevant articles were selected for initial screening. After reviewing titles and abstracts to ensure the studies focused specifically on the application of AI in higher education settings, 47 articles were retained for full-text assessment. Following a detailed evaluation of methodology and content, 24 articles met the criteria and were included in the final review.

3 Findings

A total of 24 studies published in 2024 and 2025 were included in the final analysis of this systematic review. These studies focus on examining the application of AI technologies in higher education. The results reveal a variety of AI applications, as well as the challenges and implications associated with their integration into higher education settings.

3.1 AI Application

AI can support a wide range of tasks in higher education, including:

- **Research support:** AI tools such as ChatGPT can be very helpful for research. Instead of using the typical search engines, the researchers can type the topic and question prompts to ChatGPT. ChatGPT will then respond with a well-written academic response to the prompt. If the researcher adds a request for citation to the search prompt, ChatGPT will respond with an essay completed with in-text citations. A follow-up request to prompt ChatGPT to show the reference list will produce a list of citations that match the in-text citation in the previous response [14]. Furthermore, ChatGPT can help researchers with literature review, which is the most time-consuming task in writing research [14]. Using ChatGPT can help researchers reduce the time required for literature reviews and assist in formulating research criteria and developing appropriate terminology to conduct traditional literature reviews [14]. Moreover, tools such as Research Rabbit can help build and organize research resources [7]. After resources are obtained, users can upload them into an AI program of choice, such as ChatGPT, Perplexity, or Elicit, to interact with the AI and extract information from the text, including article summaries, identifying quotes, or to prompt further questions [7]. Students conducting research who struggle with technical terms in a specific field can ask follow-up and clarification questions, interacting with the content at a level that they are comfortable with [7]. AI can also be an excellent tool assisting students in academic writing, such as improving coherence, clarity, and consistency in their work [2] as well as idea generation [5].

- **Intelligent Tutoring and Adaptive Learning:** AI can power intelligent tutoring systems, enabling prompt feedback to the user when human contact is not feasible or when interaction is not required [8, 11, 6, 18]. An example of such a platform is ALEKS [10]. Furthermore, AI can be used as a chatbot to provide personalized assistance to students, answering questions, solving problems, and providing explanations as needed [18, 10, 4]. AI can be used by instructor to adapt learning for individual students based on their preference and current level of understanding [18, 1, 22, 24]. AI system then tracks the students' progress and creates a graph to visually show where the students are in the course as students interact with the systems, AI system will then learn about the students and updates the graph accordingly, such as topics completed and topics that still need to be completed or students having difficulty achieving the learning outcome and need help[1]. Furthermore, AI systems can provide an appropriate framework to facilitate students' interactions and mutual feedback [18]. Finally, AI have been found to assist in bridging the gaps between students' learning styles and educators' teaching styles [9].

- **Data Analysis and Visualization Tools:** AI can also be used as data analysis and visualization tools, which can be used for analyzing large experimental datasets and generating visualizations [18]; among which the most widely used are Python libraries such as Pandas, Matplotlib, and Seaborn, and R with packages such as ggplot2 [10].

- **Accessibility and Remote Support:** Students who live in remote areas with the needed technology can access learning materials without leaving their community and receive support while they are learning at any time via the institution chatbot, as a consequence inequalities in education will be reduced since education will be accessible to all [1]. AI can also be used to empower learners with disabilities and support inclusive learning. Large language models can provide speech-to-text and text-to-speech capabilities to help visually impaired learners and facilitate tasks such as adaptive writing, translations, among others[16].

- **Automating Administrative Tasks:** AI enables teachers to improve courses incrementally through engagement with intelligent agents and by continually reviewing teaching quality [19, 16]. Moreover, AI can simplify the administrative workloads, including grading assignments, managing grades, organizing schedules, freeing educators to dedicate more time to teaching [18, 3, 5].

- **Human-AI Collaboration in Teaching and Learning:** It is recommended to incorporate both human intelligence and artificial intelligence

to capitalize on the strengths of both approaches [4]. Professors can use AI to create syllabi, tests, and case studies, and a chatbot can be used to explain concepts and support students in their learning, helping to clarify difficult tasks. For example, Carnegie Learning implements LiveHint AI to support math students [7]. AI power simulation system can be used to assist with creating realistic and controlled clinical scenarios [7]. Furthermore, AI can be used to create playful learning experiences that result in increased motivation and engagement among students through adaptive educational games [18]. Augmented and virtual reality powered by AI can create engaging learning environments allowing students to experience practical situations in controlled environments [18].

- **Institutional Planning and Resource Optimization:** AI can be utilized for institutional data analysis to identify problems and opportunities, thereby supporting administrators in making informed, strategic, long-term planning decisions [18]. Furthermore, AI can help institution better manage their resource by evaluating the allocation, use of resources, and recommending optimal resource distribution [18].

3.2 Challenges

There has been debate about the use of AI in higher education due to concerns with academic integrity [8, 4]. Privacy is another concern, as some AI systems may capture data and interactions through data-sharing capabilities. Another challenge is the excessive reliance on AI, which may undermine critical thinking [3]. Moreover, many institutions lack regulations and policies to ensure the responsible and ethical use of AI [4, 22].

AI assessment consistency is also debated. One study on the use of AI as an assessment tool found that its consistency compared to human expert evaluation depends on several factors, including the specific context, nature of the assessment tasks, and level of expertise required for evaluation [6]. Specifically, AI generally awarded higher scores than human evaluations in image-based exams, but both show moderate to high consistency [6]. It is also observed that there is low consistency between instructor scores and AI scores in video format exam evaluations, demonstrating AI's limitations in assessing student performance in exams conducted in video format or complex visual materials [6]. However, there is an excellent level of agreement between instructor scores and AI scores in test exams that involve multiple choice and true-false questions, demonstrating the effectiveness and reliability of using artificial intelligence when evaluating test exams using multiple choice and true-false questions [6]. Finally, integrating AI and leveraging the use of AI in higher education faces challenges due to other factors such as technological barriers, lack of training

and professional development [12], inadequate access to AI tools [23], resistance to change [15], and limited institutional leadership support [8, 17]. Furthermore, concerns have been raised that the use of ChatGPT may weaken peer interactions and leadership skills among students [23].

3.3 Recommendations

AI offers significant benefits, but its adoption poses challenges in higher education. Below are the recommendations for a successful and responsible implementation of AI in higher education:

Higher education institutions should start by enhancing their technology infrastructure, increasing funding, and developing internal technology expertise. Additionally, they should continually gather feedback from stakeholders, such as students and faculty, to further refine the AI system and meet institutional needs [17]. The successful integration of AI requires ongoing adaptation to emerging technologies and continuous evaluation of their impact [17].

Institutions should also upskill faculty and staff in AI and related technologies [17]. There should be collaboration between public and private organizations to exchange resources, expertise, and best practices for implementing AI [17]. Furthermore, it is crucial for universities to implement robust guidelines and oversight mechanisms to ensure the ethical use of AI [17, 24]. AI policies are not developed only for instructors and students, but also should include and provide guidance for other stakeholders such as administrators, learning designers, librarians, researchers, IT support staff, and registrar [1].

Furthermore, higher education needs to reform its curriculum in teaching approach, learning, and assessment design to adapt to the growth in AI educational technologies [6, 4]. Instead of focusing on having students memorize or repeat information, the curriculum should evaluate and apply the content to help students think critically, solve real problems, make comparisons, and apply knowledge to real-life situations or personal experiences [4]. Learning goals and assessments should also aim for authenticity, creativity, and include experimental or personal experiences [4]. Institutions should also prepare students for the future job market by starting to teach skills that are closely related to AI technologies [23].

Lastly, we develop a sequence of steps to help higher education institutions implement AI effectively, as indicated in Figure 2.

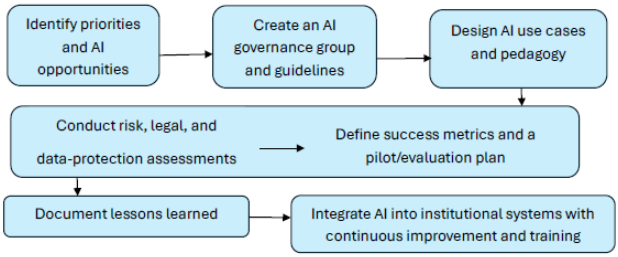

Figure 2: Sequence of steps for implementing AI in higher education

4 Conclusion

In conclusion, AI is transforming higher education, from streamlining faculty administrative tasks and facilitating research to creating an adaptive, engaging learning environment tailored to individual needs. Although these benefits come with challenges, including ethical concerns and resistance to change, technological barriers, and a lack of institutional support. Overall, AI is a technology, and by itself, it is neither inherently good nor bad; its impact depends on how it is used and governed. This highlights the need to implement policies that ensure the ethical use of AI in higher education institutions. Institutions must therefore establish clear policies and provide ethics training alongside technical instructions to ensure both students and faculty understand not only how to use AI tools but also how to use them responsibly.

References

[1] Mohamed Ally and Sanjaya Mishra. "Policies for Artificial Intelligence in Higher Education: A Call for Action". In: *Canadian Journal of Learning and Technology (CJLT)* 50.3 (2024).

[2] Gifty Edna Anani, Ernest Nyamekye, and Daniel Bafour-Koduah. "Using artificial intelligence for academic writing in higher education: the perspectives of university students in Ghana". In: *Discover Education* 4 (2025), p. 46. DOI: 10.1007/s44217-025-00434-5. URL: https://doi.org/10.1007/s44217-025-00434-5.

[3] Imre Bende. "Preparedness for Artificial Intelligence in Education". In: *Acta Didactica Napocensia* 17.2 (2024), pp. 29–36. DOI: 10.24193/adn.17.2.2.

[4] Lorraine Bennett and Ali Abusalem. "Artificial Intelligence (AI) and its Potential Impact on the Future of Higher Education". In: *Athens Journal of Education* 11.3 (2024), pp. 195–212. DOI: 10.30958/aje.11-3-2.

[5] Jorge Cordero, Jonathan Torres-Zambrano, and Alison Cordero-Castillo. "Integration of generative artificial intelligence in higher education: best practices". In: *Education Sciences* 15.1 (2025), p. 32. DOI: 10.3390/educsci15010032. URL: https://doi.org/10.3390/educsci15010032.

[6] Tugra Karademir Coskun and Ayfer Alper. "Evaluating the Evaluators: A Comparative Study of AI and Teacher Assessments in Higher Education". In: *Digital Education Review* (2024).

[7] Julie Eng et al. "AI Beliefs and Practices in Community College Classrooms". In: *Inquiry: The Journal of the Virginia Community Colleges* 28.1 (2025).

[8] Adronisha T. Frazier. "Exploring the Dynamics of Artificial Intelligence in Higher Education". In: *AI in Higher Education* (2025).

[9] Azman Hakimi et al. "The social impact of artificial intelligence chatbots on college students". In: *International Journal of Evaluation and Research in Education* 14.1 (2025), pp. 10–16. DOI: 10.11591/ijere.v14i1.29469. URL: https://doi.org/10.11591/ijere.v14i1.29469.

[10] Elizeth Mayrene Flores Hinostroza et al. "Linear regression model to predict the use of artificial intelligence in experimental science students". In: *International Electronic Journal of Mathematics Education* 20.1 (2025). DOI: 10.29333/iejme/15736.

[11] Thiti Jantakun, Kitsadaporn Jantakun, and Thada Jantakoon. "Bibliometric Analysis of Artificial Intelligence in STEM Education". In: *Higher Education Studies* 15.1 (2025). DOI: 10.5539/hes.v15n1p69.

[12] Zhang Jin, S. B. Goyal, and Anand Singh Rajawat. "The informational role of artificial intelligence in higher education in the new era". In: *Procedia Computer Science* 235 (2024), pp. 1008–1023. DOI: 10.1016/j.procs.2024.04.096. URL: https://doi.org/10.1016/j.procs.2024.04.096.

[13] Nested Knowledge. *PRISMA Flow Diagram: How publications 'flow' through the updated PRISMA 2020 process and chart*. Retrieved July 20, 2025. 2022. URL: https://about.nested-knowledge.com/2022/07/22/how-publications-flow-in-prisma-2020/.

[14] Gary Lieberman. "The Use and Detection of AI-Based Tools in Higher Education". In: *Journal of Instructional Research* 13 (2024), pp. 70–80.

[15] Jamie Magrill and Barry Magrill. "Preparing Educators and Students at Higher Education Institutions for an AI-Driven World". In: *SoTL in Process* (2024).

[16] Dana-Kristin Mah and Nele Groß. "Artificial intelligence in higher education: exploring faculty use, self-efficacy, distinct profiles, and professional development needs". In: *International Journal of Educational Technology in Higher Education* 21 (2024), p. 58. DOI: 10.1186/s41239-024-00490-1. URL: https://doi.org/10.1186/s41239-024-00490-1.

[17] Abdulla-All Mijan, Md Rabiul Hasan, and Mehedi Hasan. "AI and academia: Navigating the adoption of artificial intelligence in universities". In: *International Journal of Technology in Education and Science (IJTES)* 9.1 (2025), pp. 54–65. DOI: 10.46328/ijtes.602.

[18] Juri Evelyn Nuñez Portilla et al. "Systematic Review: Artificial Intelligence (AI) in Education 4.0". In: *Journal of Educators Online* (2024).

[19] Elkin Arturo Betancourt Ramirez and Juan Antonio Fuentes Esparrell. "Artificial Intelligence (AI) in Education: Unlocking the Perfect Synergy for Learning". In: *Educational Process: International Journal* 13.1 (2024), pp. 35–51. DOI: 10.22521/edupij.2024.13.1.3.

[20] Cole Stryker and Eda Kavlakoglu. *What is artificial intelligence (AI)?* Retrieved July 19, 2025. 2024. URL: https://www.ibm.com/think/topics/artificial-intelligence.

[21] Tableau. *What is the history of artificial intelligence (AI)?* Retrieved July 19, 2025. 2023. URL: https://www.tableau.com/data-insights/ai/history#ai-birth.

[22] Hui Wang et al. "Generative AI in higher education: Seeing ChatGPT through universities' policies, resources, and guidelines". In: *Computers and Education: Artificial Intelligence* 7 (2024), p. 100326. DOI: 10.1016/j.caeai.2024.100326. URL: https://doi.org/10.1016/j.caeai.2024.100326.

[23] Sualeha Zafar, Farzana Shaheen, and Javaria Rehan. "Use of ChatGPT and generative AI in higher education: opportunities, obstacles and impact on student performance". In: *iRASD Journal of Educational Research* 5.1 (2024), pp. 1–12. DOI: 10.52131/jer.2024.v5i1.2463. URL: https://doi.org/10.52131/jer.2024.v5i1.2463.

[24] Abdulrahman M. Al-Zahrani and Talal M. Alasmari. "Exploring the impact of artificial intelligence on higher education: The dynamics of ethical, social, and educational implications". In: *Humanities and Social Sciences Communications* 11 (2024), p. 912. DOI: 10.1057/s41599-024-03432-4. URL: https://doi.org/10.1057/s41599-024-03432-4.

The Illusion of Diversity: Mapping Homogeneity Across Generative AI Systems*

Hayden Eddy, Smera Shrestha, Nan Sun
Computer and Information Sciences
Washburn University
Topeka, KS 66621
{Hayden.Eddy,Smera.Shrestha,Nan.Sun}@Washburn.edu

Abstract

Large language models (LLMs) often present themselves as distinct systems, yet their outputs frequently converge. This study examines when and why such homogeneity emerges by evaluating five LLMs across a two-axis, four-quadrant prompt framework that varies cognitive orientation and linguistic specificity. Using 2,000 responses generated under isolated and conversational protocols, we measure overlap with both TF–IDF and sentence-embedding cosine similarity. Results show uniformly high semantic convergence across models, with logical and specific prompts producing the strongest alignment and creative, vague prompts generating the most variation. Isolated prompts also yield more homogeneous outputs than multi-turn chats. While lexical redundancy varies modestly by platform, semantic similarity remains consistently high. These findings indicate that LLMs often produce the same underlying ideas despite surface differences, highlighting structural limits on output diversity and implications for creativity, bias, and LLM-powered research.

1 Introduction

Large Language Models (LLMs) now shape how people write, learn, and search. Such systems are marketed as distinct, yet prior work suggests they often produce convergent outputs in both style and meaning [6], [9]. Studies also document domain-level homogenization and stable biases and stances, which suggest that even when responses are paraphrased, they often still convey essentially the same underlying ideas [5], [4], [7]. These patterns raise a basic question: when given the same task, do leading models truly diverge, or do they land on the same core ideas? To address this, we compare multiple LLMs across a two-axis prompt framework that varies cognitive orientation and linguistic specificity, and we test isolated versus continuous interactions. We evaluate similarity with TF–IDF (lexical) and embeddings (semantic).
We set out to answer the following questions:

- **RQ1**: To what extent do LLMs exhibit semantic and lexical homogeneity overall, both within the same model (in-group) and across different models (out-group)?
- **RQ2**: Does the degree of similarity among LLM outputs vary across different prompt types, such as creative, logical, factual, or subjective prompts?
- **RQ3**: Do some LLM platforms produce more homogeneous responses than others, or is convergence consistent across systems?
- **RQ4**: How does conversational context influence similarity? Specifically, do multi-turn chats lead to more convergence compared to isolated single-turn responses?
- **RQ5**: Do different analytical techniques (semantic similarity and lexical similarity) offer consistent assessments of homogeneity, or do they reveal different layers of overlap?

Clarifying these patterns is essential for understanding how genuine diversity and meaningful choice exist among today's leading LLMs. If systems that appear distinct consistently produce comparable ideas, users may be encountering an illusion of variety rather than true model-level differentiation. Identifying where and why these overlaps occur can inform model development aimed at preserving diversity and guide researchers, practitioners, and policymakers in interpreting AI-generated text in creative, analytical, and professional contexts.

2 Literature Review

2.1 Homogeneity, Reinforcement, and Cultural Drivers

Recent work shows that LLM outputs often converge in both style and meaning. Homogeneity persists even after assistance is removed [6], appears as creative convergence across models [9], and surfaces in narrative "echoes" that recur across generations and systems [10]. Once a framing is established, models tend to repeat it, narrowing variation over time; assisted ideation similarly becomes more fluent yet less diverse [1], with AI stories showing stable scaffolds relative to human crowd stories [2]. Convergence also reflects socio-cultural regularities: models exhibit persistent gender stereotyping and political leanings [4], [7], creative outputs tend to pull toward dominant stylistic norms [9], and applied writing such as marketing copy becomes more uniform under AI use [5]. These patterns align with system-level monoculture risks tied to shared components and overlapping data [3].

2.2 Quantifying Similarity: Semantic and Lexical Approaches

TF-IDF similarity captures surface repetition, while embeddings capture conceptual alignment. Metric comparisons help interpret clustering and dispersion [8]. Recurring plot structures despite varied wording underscore the need for semantic evaluation [10]. Evidence from ideation and creative tasks shows convergence in both style and meaning [1], [9], motivating the use of both measures.

2.3 Integrated Themes in Existing Research

Across literature, three mechanisms consistently explain why LLM outputs converge. Architectural and data overlap drives system-level monoculture risks and helps account for cross-model clustering and recurring narrative structures [3], [9], [10]. Contextual reinforcement narrows variation once a framing is set, homogeneity persists beyond immediate assistance, and ideation becomes more fluent but less diverse [6], [1]. Models are aligned to similar socio-cultural norms, their underlying biases and stances tend to converge across prompts, producing especially uniform outputs in applied writing tasks [4], [7], [5]. Methodologically, we assess sameness at two levels: *lexical* (using TF–IDF) and *semantic* (using sentence embeddings). Comparing how these two metrics behave helps us distinguish clustered response patterns from more dispersed ones [8].

2.4 How This Study Extends Prior Research

This study extends prior work on LLM homogeneity in three main ways. First, we introduce a two-axis prompt framework (logical vs. creative, specific vs. vague) and systematically apply it across five leading LLM platforms rather than focusing on a single model or narrow task. Second, we compare outputs generated under both isolated and conversational conditions, showing how interaction style shapes homogeneity within and across models. Third, we jointly analyze lexical similarity (TF–IDF) and semantic similarity (sentence embeddings) over a controlled dataset of 2,000 responses, demonstrating that apparently diverse wording can mask strong convergence at the level of ideas. Together, these design choices provide a more structured and comparative view of the "illusion of diversity" across current LLM ecosystems.

3 Methodology

3.1 Research Framework

The study began by defining a prompt typology that captured both cognitive orientation and wording. A single logical–creative scale was too limited, so we adopted a two-axis, four-quadrant framework: the horizontal axis ranges from logical reasoning to creative inference, and the vertical axis ranges from highly specific to vague, open-ended phrasing (Figure 1).

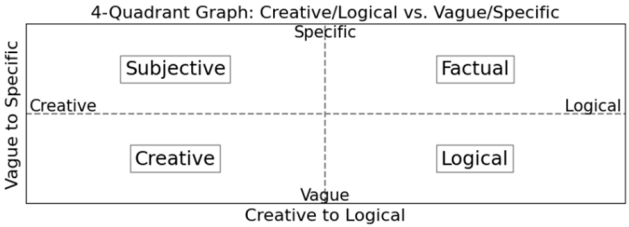

Figure 1: Two-axis prompt framework.

Crossing these axes yielded four prompt categories: *Factual* (logical/specific), *Logical* (logical/vague), *Subjective* (creative/specific), and *Creative* (creative/vague), ensuring balanced coverage of reasoning style and specificity. Prompts were developed iteratively (brainstorming, refinement, pilot testing, and classification), then reduced to remove redundancy and ensure clear quadrant fit. The final set included 40 prompts, evenly split across categories. During data collection, prompts were rarely adjusted and only when needed to prevent unintended outputs and maintain format consistency. The full prompt

list is available in our public repository at:
https://github.com/Hayd3ne/IllusionOfDiversity

3.2 Models and Interaction Protocols

We selected five LLMs spanning different architectures, training philosophies, and organizational contexts: *ChatGPT-5*, *DeepSeek-V3.1-Exp*, *Gemini 2.5 Flash*, *Llama 4*, and *Grok 4*, covering both proprietary and open-source ecosystems. To test conversational context, we used two interaction protocols: Researcher 1 submitted each prompt in a new chat session (isolated), while Researcher 2 submitted the same prompts sequentially within a single conversation (contextual), allowing prior turns to influence later responses.

Each researcher submitted each prompt to each model five times, resulting in repeated samples under both isolated and contextual conditions, yielding a total of 2,000 responses (40 prompts $\times$ 5 models $\times$ 5 trials $\times$ 2 researchers). All models were accessed through standard public interfaces with default settings; parameters such as temperature, creativity level, and output style were not modified. API access and developer modes were intentionally avoided to ensure that outputs reflected typical user-facing model behavior rather than customized configurations.

3.3 Data Collection and Storage

All responses were stored in a structured Microsoft Excel dataset. Each entry included the full model output along with metadata such as prompt text, model name, prompt category, researcher identifier (Researcher 1 or Researcher 2), trial number, and date of collection. This organization ensured that every response was traceable to its experimental condition.

For semantic analysis, each response was encoded using a standardized sentence-transformer embedding model (all-mpnet-base-v2). Embedding vectors, word counts, and character counts were stored directly in the dataset to streamline similarity computations and ensure reproducibility. The dataset was reviewed for accuracy and formatting consistency before being finalized for analysis.

3.4 Similarity Measures

To evaluate how similar two responses were, we used cosine similarity on two different vector representations: a TF–IDF bag-of-words representation to capture lexical overlap, and a sentence embedding representation to capture semantic similarity. Using both allows us to separate similarity in wording from similarity in underlying ideas.

For lexical similarity, we represent each response using TF–IDF (term frequency–inverse document frequency). TF–IDF counts how often each word appears in a response (term frequency) but down-weights words that are very common across all responses (inverse document frequency). As a result, generic words like "the" or "and" receive low weight, while more informative words like "algorithm" or "bias" receive higher weight. Each response becomes a TF–IDF vector, and we compute cosine similarity between these vectors. High cosine similarity in this space indicates that responses use very similar words and phrases, making TF–IDF a natural way to measure surface-level lexical similarity.

For semantic similarity, each response is encoded as a high-dimensional vector using a sentence-transformer embedding model (see Section 3.3). These embeddings are trained so that texts with similar meanings are mapped to nearby points in the vector space, even if they use different words. We again use cosine similarity, this time between embedding vectors, to quantify how close two responses are in meaning. High cosine similarity here indicates that responses express very similar ideas, even if their wording differs.

Cosine values range from -1 to 1, but in practice both TF–IDF and embedding vectors for our data produce values largely in the positive range. For our purposes, values above $\frac{\sqrt{2}}{2}$, or 0.708, are considered highly similar.

4 Results

4.1 RQ1: Overall Homogeneity, In-Group vs. Out-Group Similarity

We assessed whether responses are more consistent within a shared prompt context than across prompts by comparing cell-level mean cosine similarities (cells are defined as Prompt × Model × Researcher) using TF–IDF and embeddings. Within-cell similarity was much higher than across prompts for both methods: TF–IDF ($M = 0.3734$ vs. 0.0264; $\Delta = 0.3470$; 95% CI [0.3283, 0.3667], permutation $p = 0.0002$) and embeddings ($M = 0.8531$ vs. 0.1002; $\Delta = 0.7529$; 95% CI [0.7393, 0.7660], permutation $p = 0.0002$).

Table 1: In-Group vs. Out-Group Similarity

Group	TF-IDF Mean	TF-IDF std dev	Embedding Mean	Embedding std dev
In-Group	0.3734	0.2016	0.8531	0.1297
Out-Group	0.0264	0.0215	0.1002	0.0532
Difference (In – Out)	0.3470	—	0.7529	—

4.2 RQ2: Similarity by Prompt Type

We tested whether prompt structure affects homogeneity by comparing within-cell cosine similarities across four prompt types. Lexically (TF–IDF), similarity differed by type (Kruskal–Wallis $H = 28.80$, $p = 2 \times 10^{-6}$): Specific Logical ($M = 0.6759$) > Vague Logical (0.6196) > Specific Creative (0.5529) and Vague Creative (0.5634). Semantically (embeddings), differences were larger ($H = 82.02$, $p < 10^{-6}$): Vague Logical (0.9037) and Specific Logical (0.8986) were highest, with lower similarity for Specific Creative (0.8397) and especially Vague Creative (0.7705). Thus, logical prompts produce the strongest lexical and semantic convergence.

Table 2: Within-Cell Similarity by Prompt Type (TF–IDF & Embeddings)

Prompt Type	TF–IDF Mean	TF–IDF SD	Embedding Mean	Embedding SD
Specific Logical	0.6759	0.1998	0.8986	0.1323
Vague Logical	0.6196	0.1673	0.9037	0.0674
Specific Creative	0.5529	0.1868	0.8397	0.0982
Vague Creative	0.5634	0.1791	0.7705	0.1548

4.3 RQ3: Similarity by Model

We tested platform-level homogeneity by comparing within-cell cosine similarity across five LLMs. TF–IDF similarity differed by model (Kruskal–Wallis $H = 51.65$, $p < 10^{-6}$), with the highest overlap for DeepSeek ($M = 0.6922$) and Gemini (0.6663) and lower values for Grok (0.5672), ChatGPT (0.5618), and Llama (0.5273). Embedding similarity did not differ significantly ($H = 5.63$, $p = 0.228$); all models showed uniformly high alignment (≈ 0.83–0.87).

Table 3: Within-Cell Similarity by Model (TF–IDF & Embeddings)

Model	TF–IDF Mean	TF–IDF SD	Embedding Mean	Embedding SD
DeepSeek	0.6922	0.1636	0.8653	0.1217
Gemini	0.6663	0.1629	0.8469	0.1330
Grok	0.5672	0.1918	0.8678	0.1172
ChatGPT	0.5618	0.1800	0.8561	0.1074
Llama	0.5273	0.1939	0.8295	0.1594

4.4 RQ4: Learned vs. Non-Learned Behavior (Interaction Protocol)

We tested whether conversational context affects homogeneity by pairing cells with identical Prompt × Model and comparing protocols using Wilcoxon signed-rank tests (N = 200 matched pairs). The isolated protocol yielded higher lexical similarity (Δ = +0.0426; median Δ = +0.0423; 95% CI [0.0201, 0.0612]; p = 4.6e-5). Semantic similarity was slightly higher under the isolated protocol (Δ = +0.0405; median Δ = +0.0155; 95% CI [0.0060, 0.0275]; p = 4.0e-6), indicating that asking prompts in isolation produces more homogeneous responses than posing them within an ongoing conversation.

Table 4: Protocol Comparison (Isolated—Continuous)

Representation	Mean Difference	Median Difference	95% CI (Median)	Wilcoxon p-value
TF–IDF	+0.0426	+0.0423	[0.0201, 0.0612]	0.000046
Embeddings	+0.0405	+0.0155	[0.0060, 0.0275]	0.000004

4.5 RQ5: Embeddings vs. TF–IDF

Finally, we compared the similarity values produced by embeddings and TF–IDF within each cell to assess whether the two metrics capture similar patterns. Across all 400 cells, embeddings consistently produced higher similarity scores than TF–IDF (mean difference = +0.2502, median = +0.2422, 95% CI [0.2167, 0.2635], p < 1e-6). The two measures were moderately correlated (Spearman ρ = 0.6235), demonstrating shared trends but different sensitivities to lexical versus conceptual overlap.

Table 5: Paired Method Comparison (Embeddings—TF–IDF)

Metric	Value
Mean(Embeddings)	0.8531
Mean(TF-IDF)	0.6029
Mean Difference	+0.2502
Median Difference	+0.2422
95% CI (Median Difference)	[0.2167, 0.2635]
Wilcoxon p-value	<0.000001
Spearman ρ	0.6235

Embeddings capture deeper semantic convergence, while TF–IDF captures lexical repetition; both reveal consistent homogeneity patterns.

5 Discussion

Across analyses, responses were much more similar within the same cell than across prompts, with the strongest effects in embedding space. This indicates that when the task is fixed, models converge on a shared interpretation even when wording varies. Prompt structure also mattered: logical prompts produced the highest convergence (especially semantically), while creative prompts, particularly vague ones introduced more variance. Overall, task constraints appear to narrow the solution space, and "different phrasings" often express the same idea.

Platform and interaction effects added nuance. Semantic similarity was uniformly high across all five models, while lexical overlap varied, suggesting surface-style differences on top of shared conceptual cores. Interaction protocol also mattered: isolated single-turn prompting yielded slightly higher homogeneity than continuous conversation, consistent with a stabilizing effect when context is minimized. Embedding similarity was consistently higher than TF–IDF, and the two measures were only moderately correlated, implying they capture different aspects of overlap (conceptual alignment vs. word-level repetition). Taken together, LLM homogeneity appears multi-layered, driven primarily by task framing and broadly consistent across platforms.

These results imply that users may encounter less diversity of ideas than platform variety suggests. For researchers using LLMs for ideation, simulation, or literature generation, apparent model differences may mask deeper conceptual uniformity. In applied settings (writing assistance, analysis, content generation), consistency can support standardization but may also limit the range of viewpoints or alternatives surfaced by default.

Practically, users can reduce sameness by varying task framing along the two prompt axes (e.g., shifting specificity and logical constraint), explicitly requesting multiple distinct perspectives (e.g., "Give three contrasting approaches and explain how they differ"), and adding constraints such as audience, tone, or underrepresented viewpoints to push outputs beyond default "safe" answers. Intentional human variation—editing, remixing, and synthesizing across prompts, sessions, or models—can further counteract uniformity, even if underlying similarities remain.

Ethical risks follow from convergence: shared biases, cultural assumptions, or dominant perspectives may be reproduced consistently and at scale, and high semantic alignment can persist beneath paraphrases. Convergence can also reinforce errors when repeated generations echo the same inaccurate claim, which may be mistaken for evidence of correctness. These concerns highlight the need to audit outputs across prompt types, use counter-prompts to elicit alternative framings, and be cautious when homogeneity could obscure bias or error.

6 Limitations and Future Research

Several limitations should be acknowledged. First, we analyzed five models under default public-interface settings, so the findings reflect a snapshot of current systems and may shift with new releases, alternative parameters, or API configurations. Second, the scope is English and a curated set of 40 English prompts centered on explanation, opinion, and short-form creative writing; homogeneity could differ in multilingual contexts, longer outputs, or domain-specialized tasks. Third, similarity is measured with one embedding model alongside TF-IDF, so results may vary with alternative vectorizers or evaluation schemes.

Future research could expand this work by examining homogeneity across additional languages, technical or domain-specific tasks, and newer or architecturally distinct model families. Investigating how temperature, sampling strategies, system prompts, or fine-tuning methods influence similarity would further clarify the role of model configuration in producing convergent outputs. Long-form conversational studies could also reveal how homogeneity evolves over extended interactions, while ensemble prompting or multi-model workflows may shed light on strategies for mitigating convergence when diversity of ideas is desired. Together, these directions can deepen understanding of how LLMs generate meaning and how their tendency toward sameness can be both leveraged and moderated.

References

[1] Barrett R Anderson, Jash Hemant Shah, and Max Kreminski. "Homogenization Effects of Large Language Models on Human Creative Ideation". In: *Creativity and Cognition*. C&C '24. ACM, June 2024, pp. 413–425. DOI: 10.1145/3635636.3656204. URL: http://dx.doi.org/10.1145/3635636.3656204.

[2] Nina Beguš. "Experimental narratives: A comparison of human crowdsourced storytelling and AI storytelling". In: *Humanities and Social Sciences Communications* 11.1 (Oct. 2024). ISSN: 2662-9992. DOI: 10.1057/s41599-024-03868-8. URL: http://dx.doi.org/10.1057/s41599-024-03868-8.

[3] Rishi Bommasani et al. *Picking on the Same Person: Does Algorithmic Monoculture lead to Outcome Homogenization?* 2022. arXiv: 2211.13972 [cs.LG]. URL: https://arxiv.org/abs/2211.13972.

[4] Hadas Kotek, Rikker Dockum, and David Sun. "Gender bias and stereotypes in Large Language Models". In: *Proceedings of The ACM Collective Intelligence Conference*. CI '23. ACM, Nov. 2023, pp. 12–24. DOI: 10.1145/3582269.3615599. URL: http://dx.doi.org/10.1145/3582269.3615599.

[5] Chaoran Liu, Tong Wang, and S. Alex Yang. *Generative AI and Content Homogenization: The Case of Digital Marketing. SSRN Working Paper*. Available at SSRN. July 26, 2025. DOI: 10.2139/ssrn.5367123. URL: https://ssrn.com/abstract=5367123.

[6] Qinghan Liu et al. *When ChatGPT is gone: Creativity reverts and homogeneity persists*. 2024. arXiv: 2401.06816 [cs.CL]. URL: https://arxiv.org/abs/2401.06816.

[7] Pagnarasmey Pit et al. *Whose Side Are You On? Investigating the Political Stance of Large Language Models*. 2024. arXiv: 2403.13840 [cs.CL]. URL: https://arxiv.org/abs/2403.13840.

[8] Chantal Shaib et al. *Standardizing the Measurement of Text Diversity: A Tool and a Comparative Analysis of Scores*. 2025. arXiv: 2403.00553 [cs.CL]. URL: https://arxiv.org/abs/2403.00553.

[9] Emily Wenger and Yoed Kenett. *We're Different, We're the Same: Creative Homogeneity Across LLMs*. 2025. arXiv: 2501.19361 [cs.CY]. URL: https://arxiv.org/abs/2501.19361.

[10] Weijia Xu et al. "Echoes in AI: Quantifying lack of plot diversity in LLM outputs". In: *Proceedings of the National Academy of Sciences* 122.35 (Aug. 2025). ISSN: 1091-6490. DOI: 10.1073/pnas.2504966122. URL: http://dx.doi.org/10.1073/pnas.2504966122.

Put Everything Together: Reinforce Binary Knowledge in Enterprising Networking Course*

Zhengrui Qin, Sheng Chai
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, MO 64468
{zqin,schai}@nwmissouri.edu

Abstract

In recent decades, American universities have attracted a large number of international students, particularly in master's programs. These students often come from diverse academic backgrounds. In the MIS (Master of Information Systems) program at Northwest Missouri State University, the majority of students are international, with backgrounds in computer science, engineering, business, and other fields. As a result, some fundamental computer science concepts may be entirely new to a portion of the cohort. In this paper, we investigate the comprehension level of binary knowledge among students enrolled in the *Enterprise Networking* course. Several foundational topics in the course rely on an understanding of binary concepts; however, nearly one third of the students had no prior exposure to them. We argue that, in addition to teaching these topics sequentially according to the textbook, it is beneficial to offer a final integrative review session that brings all related concepts together. Such an overview helps students develop a broader conceptual framework, enabling understanding of one topic to support comprehension of the others. To evaluate the effectiveness of this approach, we administered a series of surveys. The results indicate that a dedicated review session on binary knowledge is pedagogically valuable.

1 Introduction

American universities have long been recognized as pioneers in modern higher education and continue to attract large numbers of international students [1, 4]. However, international students often arrive with diverse educational backgrounds and varying levels of prior knowledge, which can create unique challenges for university instructors. These differences may require additional effort to bridge gaps in foundational concepts, adapt teaching methods, and ensure that all students can follow the course material effectively. As a result, teaching a class with international students demands flexibility, patience, and inclusive strategies to support every student's success [2, 5].

In this paper, we present our experience in teaching a graduate-level course, *Enterprise Networking*, with a particular focus on a fundamental computer science concept: binary knowledge. The course enrolls primarily international students from diverse academic backgrounds and disciplines, including computer science, engineering, and business. Several course topics require an understanding of binary concepts, almost from the very beginning of the semester to the very end, which poses challenges for many students. In addition to teaching these topics sequentially following the textbook, we intentionally reserve one class session before the final exam week for a comprehensive review of all binary-related topics.

To evaluate the effectiveness of this instructional approach, we administered three surveys measuring students' comprehension of binary concepts at different stages of the course, observing how their understanding evolved over time. The results indicate that a dedicated review session on binary-related topics is pedagogically beneficial. To the best of our knowledge, this paper represents the first study to examine such an approach in this manner.

The rest of paper is organized as follows. Section 2 briefly describes course setup on binary-related topics in *Enterprise Networking*. Section 3 presents the questionnaires of three surveys. Section 4 shows the survey results along with our observations and analysis. Section 5 concludes the paper.

2 Course Setup on Special Topics

Enterprise Networking is a required course for the Master of Information Systems program at Northwest Missouri State University. The course covers data communications and networking protocols, with emphasis on their practical application to enterprise networking and internetworking environments. In this section, we identify the topics that rely on binary knowledge, describe how these topics are assessed, and outline our instructional plan for teaching them.

2.1 Topics Related to Binary

In this course, the following topics requires the binary knowledge:

1. Encoding/Decoding: Converting between text and ASCII.
2. MAC address: Converting MAC address between binary format and hexadecimal notation.
3. IPv4 address: Converting IP address between binary format and Dotted Decimal Notation.
4. IPv6 address: Simplifying IPv6 address from binary format to hexadecimal notation.
5. Routing: Applying mask to IP address to determine routes (AND operation).
6. Subnetting: Determining the IPv4 address of a certain host in a certain subnet.

These topics are covered across different chapters of the textbook, *Business Data Networks and Security* [3]. Specifically, Topic 1 appears in Chapter 2, Topic 2 in Chapter 5, Topics 3, 4, and 5 in Chapter 8, and Topic 6 in Chapter 9. As such, the topics are distributed over nearly three-quarters of the semester. Even though Topics 3, 4, and 5 are all within the same chapter, their coverage still extends over approximately three weeks.

2.2 Assessment

Each of the topics above is supported by dedicated assignments and evaluated in both the midterm and final exams. Here, subnetting is used as an illustrative example of how we assess each topic.

Assignment: A firm is assigned network part 137.185. It selects a 7-bit subnet part. a) Write the binary format for the 4 bytes of the IP address of the first host on the first subnet; b) Convert part a) into dotted decimal notation; c) Write the binary format for IP address of the third host on the third subnet; d) Convert part c) into dotted decimal notation.

Midterm Exam (multiple-choice): A firm is assigned the network part 134.171 and it selects a 6–bit subnet part. Then the binary format for the 4 bytes of the IP address of the second host on the second subnet is: 10000110. 10101011._____._____. The corresponding dotted decimal notation is: 134.171 ._____._____.

A) 00000010, 00000010, 2, 2.

B) 00000001, 00000001, 1, 1.

C) 00001000, 00000010, 8, 2.

D) 00000100, 00000001, 4, 1.

Final Exam (fill in blanks): A firm is assigned the network part 128.169 and it selects a 11–bit subnet part. Then the binary format for the 4 bytes of the IP address of the second host on the second subnet is: 10000000.10101001._____ ._____. The corresponding dotted decimal notation is: 128.169._____._____.

2.3 Instructional Plan

We first follow the textbook and lecture on the topics chapter by chapter, in the same order they appear in the textbook. For each topic, we walk through the procedure in detail and work through an example. Our emphasis is on teaching students the underlying logic and the steps involved in the calculations, rather than having them memorize formulas or procedures. Considering the correlations among these topics, we devote one class in the final week before the exam period to reviewing all of them together, that is, consolidating all content related to binary concepts. We believe that this comprehensive review session helps strengthen students' understanding, allowing mastery of one topic to reinforce comprehension of the others.

3 The Survey

To assess students' evolving comprehension of binary concepts, we administered three surveys: the first at the beginning of the semester, the second upon completing all chapters but before the final review session on binary knowledge, and the third after the final review session. Below are the questionnaires, respectively.

3.1 Survey 1

For the first survey, we aim to collect information about students' backgrounds and their prior knowledge of converting between binary and decimal/hexadecimal representations. The questionnaire for the first survey is as follows:

1. What is your background before coming to Northwest?
2. Do you know how to convert binary to decimal?
3. Do you know how to convert binary to hexadecimal?
4. Do you know how to convert decimal to binary?
5. Do you know how to convert hexadecimal to binary?

For the first question, we provide four choices: Computer Science, Engineering, Business, and None of above. For the other four questions, we provide three choices: "Never knew", "Used to know but forgot now", and "Used to know and still remember".

3.2 Survey 2

For the second survey, we aim to collect students' status on all topics related to binary knowledge when all chapters were finished but before the final review session. The questionnaire is as follows:

1. After taking this course, do you know how to convert binary to decimal?
2. After taking this course, do you know how to convert binary to hexadecimal?
3. After taking this course, do you know how to convert decimal to binary?
4. After taking this course, do you know how to convert hexadecimal to binary?
5. After taking this course, do you know how to convert IPV4 address from binary to Dotted Decimal Notation?
6. After taking this course, do you know how to convert MAC address from binary to hexadecimal notation?
7. After taking this course, do you know how to convert IPv6 address from binary to hexadecimal notation?
8. After taking this course, do you know how to obtain IPv4 address for a certain host in a certain subnet giving the network part?
9. After taking this course, do you know how to apply network mask to a certain IP address?

For each of the questions in the second survey, we provide the same choices: "Once knew but forgot now", and "Yes I know". The reason we exclude "Never knew" is that, based on their homework, we knew all students were once familiar with the contents.

3.3 Survey 3

For the third survey, we aimed to evaluate whether students had fully grasped the relevant concepts after the final review session and to gather their feedback on the final review session. The questionnaire is as follows:

1. After the binary review session, do you know how to convert IPV4 address from binary to Dotted Decimal Notation?
2. After the binary review session, do you know how to convert Mac address from binary to hexadecimal notation?
3. After the binary review session, do you know how to convert IPv6 address from binary to hexadecimal notation?
4. After the binary review session, do you know how to obtain IPv4 address for a certain host in a certain subnet given the network part?
5. After the binary review session, do you know how to apply network mask to a certain IP address?
6. Do you think it is worth having the binary review session?

7. Should the instructor have this review session in the future?

8. Do you have any suggestion regarding teaching the topics related to binary?

For Questions 1-5, we provide two choices: "Yes I know", and "No I still don't know". For Question 6 and 7, we provide three choices: "Yes", "No", and "Neutral". For Question 8, it is a text entry box.

4 Survey Results and Analysis

In this section, we present our survey results along with accompanying analysis. Because the three surveys were administered at different stages of the course, our goal is to examine how students' mastery of binary-related knowledge evolves over time. In particular, we aim to evaluate the effectiveness of both the regular instruction and the final review session on these topics. Twenty students participated in each of the first two surveys, and eighteen students completed the third. It is important to note that some students skipped individual questions, so the number of responses for certain questions may be one or two fewer than the total number of participants. We acknowledge that the survey is under limitation due to the small number of participants. We report the first question of the first survey separately; for all remaining questions, we present the results in a comparative format rather than survey by survey.

4.1 Students' Different Background

For the first question of the first survey, it asked students what their backgrounds were. The result is shown in Fig.1. As we can see, the students came from very diverse academic backgrounds. Interestingly, half of the students were not from computer science but from engineering, business, and other fields. Given such diverse backgrounds, it is essential to ensure that all students are adequately supported.

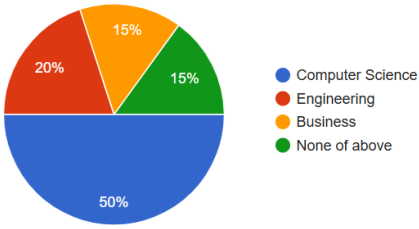

Figure 1: Students in Enterprise Networking with different backgrounds.

4.2 The Change from Survey 1 to Survey 2

Table 1 presents students' comprehension levels regarding conversions between binary and decimal/hexadecimal before the course and before the final review session, respectively. From the left column of Table 1, we observe that nearly one third of the students had no such prior knowledge before taking the *Enterprise Networking* course, which roughly aligns with the diverse backgrounds illustrated in Fig.1. After completing the course but before the final review session, the proportion of students who understood these conversions increased by at least 30%. The data also show that converting from binary to decimal is the easiest task for students (95%), whereas converting from hexadecimal to binary is the most challenging (72%).

Table 1: Comparison of students' comprehension levels of binary–decimal and binary–hexadecimal conversions at two different stages.

	Survey 1			Survey 2	
Status	never knew	forgot	still remember	forgot	still remember
b to d	30.0%	10.0%	60.0%	5.0%	95.0%
d to b	26.3%	21.1%	52.6%	15.0%	85.0%
b to h	35.0%	30.0%	35.0%	15.0%	85.0%
h to b	30.0%	30.0%	40.0%	27.8%	72.2%
b=binary; d=decimal; h=hexadecimal.					

4.3 The Results of Survey 2 and Survey 3

Table 2 presents students' comprehension levels before and after the final review session for five networking topics. The five topics listed in the left column correspond to the abbreviated forms of Questions 5–9 in Survey 2 and Questions 1–5 in Survey 3. The results clearly indicate that the final review session substantially improved student understanding. Before the final review session, students demonstrated the highest comprehension on *IPv4 to DDN* (84%) and the lowest on *IPv6 to Hex* (58%), with *IP masking* also relatively challenging (60%). Encouragingly, after the final review session, all students understood all five topics, with the exception of one student (6%) who continued to struggle with *IP masking*. These findings support our objective that a final integrative review session on correlated topics is indeed beneficial for student learning.

Fig.2 shows the results for Questions 6–7 of Survey 3. The majority of students indicated that the final review session was worthwhile (88.9%) and that it should be continued in future offerings of the course (66.7%). Notably, only one student (5.6%) responded "No" to both questions, with all remaining

Table 2: The comprehension level comparison of 5 networking topics between before and after the final review session.

	Survey 2		Survey 3	
Status	forgot	still remember	forgot	still remember
IPv4 to DDN	15.8%	84.2%	0%	100%
MAC to Hex	26.3%	73.7%	0%	100%
IPv6 to Hex	42.1%	57.9%	0%	100%
IP for subnetting	26.3%	73.7%	0%	100%
IP masking	40.0%	60.0%	5.6%	94.4%

students selecting either "Yes" or "Neutral."

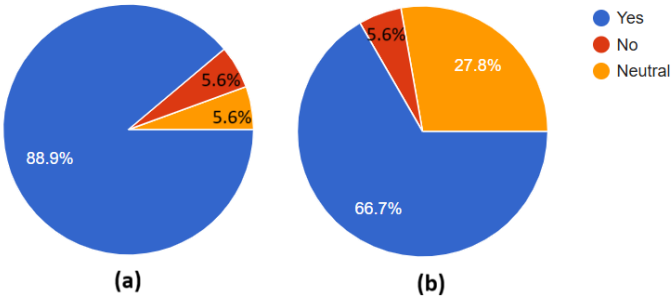

Figure 2: (a): Is it worth having the final review session? (b): Offer the final review session in the future?

For the last question in Survey 3 that asks the comments, the five responses are as follows: "It is perfect"; "Thank you, we do not need to review the concept for the final exam"; "The review was really helpful"; "The instructor is perfect, I like his patience to explain the topic"; and "The review class is very helpful". Other participants either left no comment or answered "no comment".

5 Conclusion

In this paper, we investigate how students from diverse academic backgrounds develop their understanding of binary-related concepts in *Enterprise Networking* course. Because these topics are distributed across multiple chapters of the textbook and covered over an extended period, we argue that a final integrative review at the end of the semester is pedagogically beneficial. To evaluate this approach, we administered three surveys to assess students' knowledge at dif-

ferent stages of the course. The results indicate that the final review of these related topics is indeed helpful and educationally valuable. Based on these findings, we recommend that instructors facing similar instructional contexts consider incorporating a comprehensive review session before the final exam.

References

[1] John Bound et al. "The globalization of postsecondary education: The role of international students in the US higher education system". In: *Journal of Economic Perspectives* 35.1 (2021), pp. 163–184.

[2] Chris Forlin. "Diversity and its challenges for teachers". In: *Future directions for inclusive teacher education*. Routledge, 2012, pp. 83–92.

[3] Raymond R Panko and Julia L Panko. *Business data networks and security*. Pearson Harlow, Essex, 2015.

[4] Frederick Rudolph. *The American college and university: A history*. Plunkett Lake Press, 2021.

[5] Christine E Sleeter and Jenipher Owuor. "Research on the impact of teacher preparation to teach diverse students: The research we have and the research we need". In: *Action in teacher education* 33.5-6 (2011), pp. 524–536.

Teaching Social Engineering in a Collegiate Cybersecurity Program: An Experience Report*

Tim DeClue[1] and June Middleton[2]

[1]Division of Computing and Math
Southwest Baptist University
Bolivar, MO 65613
tdeclue@sbuniv.edu

[2]Jack Henry and Associates
3725 East Battlefield Street
Springfield, MO 65809
jumiddleton@jackhenry.com

Abstract

Social engineering attacks continue to pose a significant threat to the safety of digital information. Verizon's 2025 Data Breach Investigation Report notes that a majority (53%) of data breaches include a human behavior attack vector (phishing, malicious insider, physical theft, insider error, vulnerability exploit, and/or compromised credentials). While the average 2025 data breach cost was $4.44 million globally, U.S. data breaches averaged over $10 million[4]. This paper reports on a social engineering course taught in a cybersecurity degree program at the university level. The overall objective of the course was to both teach common social engineering practices and increase student ability to recognize and prevent social engineering/human attack vectors. The report includes an examination of the curriculum used, student assignments, required research readings, and lab assignments.

1 Introduction

Social engineering attacks continue to pose a threat to the safety of digital information. A majority (53%) of data breaches involve some aspect of human behavior (phishing, malicious insider, physical theft, insider error, vulnerability exploit, and/or compromised credentials)[3], and the average U.S. data breach cost surged past $10 million[4]. This kind of challenge has existed for many years and these types of attacks are projected to continue well into the future. In response, the business community has called for more employees with advanced cybersecurity skills, universities have established undergraduate and graduate cybersecurity programs, and ABET has produced accreditation standards for undergraduate cybersecurity programs[1]. ABET's accreditation criteria requires the curriculum to include "... the study of human behavior in the context of data protection, privacy, and threat mitigation"[1]. The social engineering course detailed in this paper was meant to address this requirement. Social engineering education, here defined to be coursework that addresses the intersection between human behavior and cybersecurity practices, is an example of a crosscutting concept. The course created to address the need for graduates to have a deep understanding of the threat social engineering attacks pose was named simply "Social Engineering." The author's university established an undergraduate cybersecurity program in 2019 and received ABET accreditation in 2021. The course has been taught four times on an every-other-year basis. Seven students completed the course in the spring semester of 2025.

2 Curriculum

There was no established social engineering curriculum when the course was initially designed, so the faculty designed the course around employer's needs, a DIY-like textbook, some social engineering research, a set of labs and assignments, and a culminating capstone team assignment. Additionally, a professional social engineer at Jack Henry & Associates (JHA) was retained as a consultant in the design process to make sure academic activity retained real-world validity. The social engineer is employed by a major financial institution to perform simulated social engineering attacks on JHA clients for the purpose of discovering and addressing vulnerabilities related to human behavior/social engineering attack vectors. Each time the course was taught, the curriculum was reviewed and updated.

The text adopted for the course was "Social Engineering: The Science of Human Hacking" authored by Christopher Hadnagy[6]. The text is not a traditional academic textbook in the sense that there are no end of chapter assign-

ments, quizzes or problems. Rather, the book is organized around the fundamental skills and insights needed by professional social engineers. A strength of the text is that the author based his insights in both personal experience and peer-reviewed research.

3 Required Research Readings

To increase the rigor of the course, students were required to read several published, peer-reviewed research papers, write a review using a standardized form, and then present and discuss the significance of the research as it relates to social engineering in class sessions. One class period presented categories of research, and how to read and interpret research papers.

The papers reviewed in the spring, 2025 course are listed below. See complete citations in cited works:

1. Creating a Code of Ethics for Social Engineering in Cybersecurity: A Case Study by Abraham Alfred[2].
2. The Social Engineering Personality Framework by Sven Uebelacker and Susanne Quiel [13].
3. A Room with a Cue: Personality judgments based on offices and bedrooms by Samuel Gosling, Sei Ko, Thomas Mannarelli, Margaret Morris[5].
4. The Nature of Rapport and Its Nonverbal Correlates by Linda Tickle-Degnen and Robert Rosenthal[12].
5. Performing Social Engineering: A qualitative study of information security deceptions by Kevin Steinmetz, Alexandra Pimentel and W. Goe[11].
6. Postural Influences on the Hormone Level in Healthy Subjects: I. The Cobra Posture and Steroid Hormones by Rinad Minvaleev, A. Nozdrachev, V. Kir'yanova, and A. Ivanov[8].
7. Enhancing Information Security Literacy: The Impact of Framing Across Non-Technical Trainees by Robert Schneider[10].
8. Evaluating Truthfulness and Detecting Deception by David Matsumoto, Sung Hyi Hwang, Lisa Skinner and Mark Frank[7].
9. Consistency-based compliance across cultures. R. Cialdini, R. and S. Sills[9].

The questions on the review form were:

1. Describe the relationship of the paper to the current chapter and topic from the textbook. Be as specific as you can.
2. Describe or discuss the findings or conclusions reached by the authors of the research paper.

3. Use the idea of triangulation—increasing the credibility of information by finding it confirmed from multiple (triangulated) sources. List and describe at least two things that are discussed in the research paper and which ALSO appear in the Hadnagy text. Cite where you find the items in both the paper and your text.

4 Student Labs and Testing

Clearly, a course which studies human behavior and how to influence it should include some significant assignments which require human action and self-reflection. The following six labs were completed in the first 8 weeks of the course to prepare students for the capstone project detailed in a later section. OSINT is "Open-source Intelligence."

Lab 1 Increasing OSINT Awareness – Students were required to research themselves on common social media platforms, as well as several people search engines (PiPl, Spokeo, Truthfinder, etc.) to ascertain the extent of their digital footprint. The students were then required to remove themselves from three of the search engine's databases.

Lab 2 OSINT Collection — Students take a picture of an office and speculate on the personality of the occupant based upon the image. In step two, students are given a picture of an office for whom personality information is known. The student again speculates on the personality of the occupant before seeing the actual personality information. Finally, students use recommended tools to compile a profile for three small businesses (coffee shops were chosen arbitrarily) based upon openly available information.

Lab 3 Maltego Basics — Students are guided through a basic introduction to Maltego, a commonly used tool by social engineers for discovering, compiling, analyzing, and recording open source information.

Lab 4 Identifying Social Influence — Students first identify commonly used influence techniques in marketing and social engineering (`https://www.hooksecurity.co/phishing-email-examples`), then the students design a phishing email utilizing the techniques.

Lab 5 Performing Reconnaissance — Students focus on enhancing their observational skills by walking through a public space and then recording some basic information about the space (how many exits, how many cameras, level of security, etc.) Finally, the students take pictures of the contents of dumpsters used by the building and describe the activities they think can be supported by the pictured evidence.

Lab 6 A Simulated Phishing Campaign — Students design and launch a simulated phishing campaign using a free (limited) tool (`https://caniphish.com`). This tool limits the number of emails sent to 10, but does collect success metrics.

Each chapter had a short quiz to assess student comprehension of the subject matter. Two comprehensive tests were given; one at midterm and one as a final exam for the course. Approximately half of the tests were multiple choice questions, with the other half focusing on the analysis of descriptive scenarios. It is worth noting that ChatGPT was particularly useful in writing detailed scenarios designed to test student knowledge of social engineering attack vectors.

5 The Capstone SEVA Project

Almost every course activity prepared or supported the students in performing a social engineering vulnerability assessment (SEVA) as a capstone activity for the course. In the spring of 2025, the subject of the assessment was the county government where the university is located.

In general, government entities work well for this purpose; they have a public service mission which makes them more likely to take part in the project, and they deal with significant amounts of money and private constituent information. Social engineers consider attacking mid-size organizations like county governments for these reasons and also because county governments may be lax in funding cybersecurity training to prevent social engineering attacks.

SEVA Phase 1 (1 week) Students brainstormed and documented their ideas for assessing the social engineering vulnerability of the county. The class then met with the county clerk—the person at the county tasked with preventing cybersecurity attacks–and discussed what activities would be allowed or disallowed when the project was in motion. A memorandum of understanding (MOU) signed by both students and the county clerk represented the deliverable from this phase.

SEVA Phase 2 (3 weeks) OSINT collection on only the elected county officials began—this limitation was a specification in the MOU. Even with this limitation, however, there were 14 individuals included in the assessment. The work was divided by the three teams in the class with each team agreeing to take a subset of elected officials. Every team researched the sheriff's office by mutual agreement. The second phase deliverable was a rough draft, incomplete version of the final SEVA report.

SEVA Phase 3 (2 weeks) In this phase, each student team visited the offices of the elected officials with a pretext for the purpose of collecting non-technical OSINT. The class also orchestrated an early morning dumpster dive for the purpose of discovering OSINT in discarded rubbish. Using information from the dumpster dive, the in-person visit, and other collected OSINT, each team designed a complete social engineering attack targeted at exploiting the social engineering vulnerabilities at the county. The teams updated the rough draft SEVA report with this new information as the phase 3 deliverable.

SEVA Phase 4 (2 weeks) In the final phase, each student on the team visited the county courthouse for the purpose of collecting non-public, but innocuous information. Examples included cell numbers, home addresses, and/or non-government business cards of elected officials. The simulated phishing campaign was carried out and results compiled. The SEVA report was again updated with the results of the simulated social engineering attack, and this final report was delivered in-person to an assembly of the elected officials and their staffs.

6 Real-world Application and Course Wrap-up

Over the years, the university's social engineering courses have benefited from direct access to a local ethical social engineering subject matter expert (SME). This SME was personally trained in the psychology and advanced techniques of ethical social engineering by Chris Hadnagy, the author of the course textbook and a recognized authority in the field. Holding two ethical social engineering certifications and possessing more than a decade of industry experience, the SME provided invaluable guidance to students preparing for the social engineering vulnerability assessment (SEVA) project. In addition to offering consultation upon request, the SME delivered a comprehensive wrap-up presentation that illustrated real-world applications of course concepts. These examples were drawn from the SME's professional engagements, which included conducting vulnerability assessments for banks and credit unions nationwide.

In preparation for the SEVA project, students were encouraged to schedule video consultations with the SME to review rules of engagement and discuss the open-source intelligence (OSINT) they had gathered. These sessions facilitated the development of effective pretexts and attack vectors, such as phishing, vishing (phone elicitation), smishing, dumpster diving, and physical access attempts. The SME also provided strategic advice on adapting pretexts in real time during assessments, ensuring students understood the dynamic nature of social engineering engagements. Furthermore, professional best practices were emphasized, including critical "do's and don'ts"—for example, maintaining

possession of a "get out of jail" letter during physical access attempts and strictly adhering to the written rules of engagement to avoid unauthorized actions.

To conclude the semester, the SME delivered a one-hour presentation showcasing real-world examples of OSINT collection, tools of the trade, psychological manipulation techniques, and attack scenarios targeting financial institutions. These case studies included phishing, vishing, dumpster diving, and physical access attempts, along with their outcomes—whether successful or not. This final session extended learning beyond the capstone SEVA Project by demonstrating practical applications of theoretical knowledge. Exposure to authentic industry practices not only enhanced students' readiness for technical careers but also sparked interest in pursuing specialized roles in social engineering vulnerability assessments.

The sponsoring county was asked for some feedback and had the following comments following the SEVA report:

- "... the overall project was a success for the students and our staff alike. It gave them a great opportunity to put their skills to practice and see what that process looks like. For our organization, it was good to be reminded that even in a small rural area we are still at risk and have potential vulnerabilities."
- "As far as our cybersecurity for <county name> we have not made any changes that I am aware of as we felt what we have in place is the best that we can afford to provide. Some of our elected officials did take a look at their personal online footprint and made some changes. I also think some pay more attention to what they are throwing away and how it is being discarded."

7 Observations

Although there were only seven students in the class, the course evaluations revealed the students enjoyed the class, but felt like too little time had been allocated for collecting OSINT on the county officials. The students also did not like the amount of time outside of class required to complete the assignments. Both criticisms will be factored into the schedule the next time the course is taught. It should also be noted that the course evaluations were similar to previous versions of the course: these versions had more students and used a similar course design.

Anecdotally, the instructor observed that some students were uncomfortable presenting themselves to others with a pretext for gaining information. While this is understandable, the approach was educationally valuable in increasing

students' understanding of the practice and the approach was approved by the county government in the memorandum of understanding.

8 Summary

Social engineering poses a significant instructional challenge for academic programs. Some of the topics, like social engineering, fall outside of any traditional computing body of knowledge. This paper describes one successful approach for blending traditional academic techniques, innovative labs, and a practical, applied project to achieve an engaging course experience for teaching a challenging subject.

References

[1] ABET. *Criteria for Accrediting Computing Programs: Cybersecurity Program Criteria*. Accessed: 2025-11-29. URL: www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs.

[2] Abraham Alfred. "Creating a Code of Ethics for Social Engineering in Cybersecurity: A Case Study". In: *Proceedings of the Wellington Faculty of Engineering Ethics and Sustainability Symposium* (2022).

[3] Verizon Business. *2025 Verizon Data Breach Investigations Report*. URL: https://www.verizon.com/business/resources/reports/dbir.

[4] IBM Corporation and Poneman Institute. *IBM Cost of Data Breach Report 2025*. URL: https://www.ibm.com/reports/data-breach.

[5] Samuel D Gosling et al. "A Room with a Cue: Personality Judgments Based on Offices and Bedrooms". In: *Journal of Personality and Social Psychology, vol. 82, no. 3* (2002). DOI: https://doi.org/10.1037/0022-3514.82.3.379.

[6] Christopher Hadnagy. *Social Engineering: The Science of Human Hacking*. Reading, Massachusetts: Wiley Publishing, 2018. ISBN: 978-1-119-43338-5.

[7] David Matsumoto et al. *Evaluating Truthfulness and Detecting Deception*. Law Enforcement Bulletin, 2011. URL: https://leb.fbi.gov/articles/featured-articles/evaluating-truthfulness-and-detecting-deception.

[8] RS Minvaleev et al. "Postural Influences on the Hormone Level in Healthy Subjects: I. The Cobra Posture and Steroid Hormones". In: *Human Physiology, vol. 30* (2004). DOI: https://doi.org/10.1023/B:HUMP.0000036341.80214.28.

[9] Petia K Petrova, Robert B Cialdini, and Stephen J Sills. "Consistency-Based Compliance Across Cultures". In: *Journal of Experimental Social Psychology* 43.1 (2007), pp. 104–111.

[10] Robert Schneider. "Enhancing Information Security Literacy: The Impact of Framing Across Non-Technical Trainees". In: (2024).

[11] Kevin F Steinmetz, Alexandra Pimentel, and W Richard Goe. "Performing Social Engineering: A Qualitative Study of Information Security Deceptions". In: *Computers in Human Behavior, vol. 124* (2021). DOI: https://doi.org/10.1016/j.chb.2021.106930.

[12] Linda Tickle-Degnen and Robert Rosenthal. "The Nature of Rapport and Its Nonverbal Correlates". In: *Psychological inquiry* 1.4 (1990), pp. 285–293.

[13] Sven Uebelacker and Susanne Quiel. "The Social Engineering Personality Framework". In: *Proceedings of the IEEE Security and Trust Symposium* (2014). DOI: https://doi.org/10.1109/STAST.2014.12.

Last Day, Lasting Impact: Maximizing the Final Day of your Computer Science Course*

Mark Terwilliger, Elise Terwilliger, and Janet Jenkins
Computer Science and Information Systems
University of North Alabama
Florence, AL 35632
{mterwilliger, eterwilliger, jltruitt}@una.edu

Abstract

The final day of a college course is often underutilized, typically reduced to brief administrative remarks or exam reminders. This paper describes a structured end-of-semester reflection activity designed to make the final class meeting meaningful in computer science courses. Grounded in Science of Learning principles, the activity integrates retrieval practice, metacognitive reflection, elaboration, growth mindset, and intellectual closure. Implemented across six undergraduate courses with 103 students, survey results and student feedback indicate strong perceived benefits, including improved recall, conceptual understanding, confidence, and a sense of community.

1 Introduction

At the start of the semester, college professors are energized and prepared. Professors have reviewed syllabi, outlined goals, and returned from break ready to engage students with the course content. Students, too, begin with a clean slate and shared sense of optimism. Throughout the semester, professors invest

in lectures, assignments, and assessments, while students (ideally) commit to attending class, studying, and completing their work.

By semester's end, professors face piles of ungraded work and looming reports, while students juggle final assignments and exam prep, often feeling overwhelmed. Everyone is ready to be done, making the final class session a brief exchange: "Any questions?" — "No." — "Good luck on the final."

As university faculty members who have taught Computer Science (CS) and Mathematics courses for a combined 74 years, we were given pause by an interesting article that suggested a new purpose for that last day of class [8]. Maybe it is time to treat the last day of class as a first-class citizen. Specifically, is there something we can do on the final day of the semester that is of benefit to our students? Further, given certain common personality traits we have seen over our years of teaching, we wonder how this last day activity may be particularly helpful in the CS classroom.

To answer this question, we developed and implemented a protocol on the last day of class in our courses. This paper provides an overview of the ideas that motivated our project, along with the methodology we developed, and the results we observed.

2 Background - Science of Learning

The article, "Don't Just Fade to Black: Ending a Course With Purpose.[8]" outlines seven meaningful strategies for concluding the final day of the semester. Each strategy is grounded in one of these Science of Learning principles, highlighting the research-based rationale behind the approach:

1. Retrieval + Reflection: Encouraging students to recall the most significant concepts they learned throughout the course can lead to valuable discussions. This practice leverages the retrieval effect, which demonstrates that actively recalling information enhances memory and comprehension more effectively than passive review[4].

2. Metacognitive Awareness: By reflecting on course learning goals and discussing how topics are interconnected, students develop metacognitive awareness. This reflection helps them recognize overarching themes and the relationships between concepts[6].

3. Elaboration: Asking students to write a short note to future students—sharing advice or insights they wish they had known—provides an opportunity for elaboration through teaching. Teaching others has been shown to deepen understanding and reinforce learning[10][9].

4. Growth Mindset: Fostering a growth mindset and reinforcing self-efficacy are critical to student persistence and resilience. One activity invites students to identify something they can do now that they could not at the start of the semester, reinforcing the progress they have made[2][7].

5. Schema Construction: Students complete the prompt: "This course was really about _____" An activity like this encourages schema construction—the process of synthesizing and connecting course content into a cohesive understanding, making learning more meaningful and less fragmented[10][9].

6. Student Voice: Allocating time for open discussion gives students the chance to share feedback and reflect on their experiences. This promotes a sense of community and provides emotional closure as the course concludes.

7. Intellectual Closure: The aim here is to bring intellectual closure by revisiting early-semester questions and exploring how student understanding has developed over time. This reflection supports a sense of belonging and academic self-efficacy[1].

3 Background - CS Students and Personality Types

The Myers-Briggs Type Indicator (MBTI) describes the personality type of an individual using four categories, where each category has two traits[5]. The personality type of a person can be determined through testing designed to summarize a person's natural tendency within each of the four categories. This yields 16 possible distinct personality types.

One of the four categories is Extraversion/Introversion. It has been shown that there is a statistically significant difference in the Extravert/Introvert category between CS students and the general population of students[3]. In this study, about 50% of the CS undergrad students in their study fall into just two of the 16 categories, ISTJ or ISFJ, meaning CS students tend to prefer introversion, sensing, and judging.

As introverts (I), these students prefer to focus on their inside world versus the outside world. They take more time for reflection and prefer to think things through before acting, while their extrovert counterparts tend to act quickly, often before thinking things through. In addition, introverts usually prefer communicating through writing and processing ideas inwardly, while extroverts generally communicate through talking and will often process ideas outwardly. Therefore, CS students may need benefit from additional time to respond to questions posed to them during class by the instructor or they may

be reluctant to answer at all, which hinders building a sense of community. The need for processing time was considered in the design of the reflection portion of this project to provide proper time to think before speaking.

Students who prefer judging (J) over perceiving (P) tend to value closure and structure. They want to have control and tend to work in a methodical manner. This activity was designed to be organized and fully explained prior to the onset of the activity. As part of the design, the students would be given control over whether they wanted to share the answers from the reflective portion. The last part of the activity is designed to help the students gain a sense of closure and is detailed in the Methodology section.

4 Methodology

We found merit in all seven meaningful strategies described in Section 2. Consequently, we created a single period activity that incorporated elements of all of them. Informed by the lessons learned in Section 3, the protocol incorporated elements that first gave students time to process individually in writing, followed by a period for sharing their thoughts orally with the group.

We tested our activity in six course sections, with a total of 103 students, most of whom were computer science majors. The courses included CS1 (26 students), CS Discrete Structures (15 students), CS2 (2 sections, 10 students each), and Computer Organization and Assembly Language (2 sections, 27 and 15 students), and were taught by three different instructors. Five of these sections were 75-minute class periods, while the CS1 section was a 50-minute class period.

The class time for the 75-minute sections was arranged as follows:

1. Review for final exam. Discuss an exhaustive list of all topics covered during the semester. 10 minutes
2. Explain the activity and discuss the Informed Consent form. 5 minutes
3. Using their computer, students individually fill out the Course Reflection Worksheet and then upload the document as an assignment. 20 minutes
4. Instructor goes through the worksheet questions with class, soliciting input from as many students as possible. Instructor takes notes. 25 minutes
5. Students take wrap-up survey, ranking how each of the Science of Learning principles felt meaningful/impactful by doing this activity. 15 minutes

The Course Reflection Worksheet looked like the following:

1. Write down the three most important things you learned in this course.
2. How are some of the concepts from this course interconnected? Give examples.
3. Write a short note to a future student taking the course with any thoughts or advice you might have. Some possible helpful prompts are "Here's what I wish I'd known..." or "This helped me succeed...", but it could be anything valuable you'd like to share with them about the course.
4. Identify at least one thing you couldn't do at the beginning of the course that you can do now. Choose something that you struggled a bit to learn or to understand about the course material or about yourself. It could be a course topic, a skill, a mindset, or simply an area of increased confidence.
5. In one sentence, describe what this course is really all about.
6. a) What are you taking away from this course? b) What surprised you about this course? c) What challenged you about this course? d) What do you wish the instructor would have done differently? e) What do you wish you had done differently?
7. What sort of project or task has this course prepared you to tackle next?

The Wrap-Up Survey looked like the following:

1. The course summary discussion helped to strengthen my recall and understanding of what was learned during the semester.
2. The course summary discussion helped me to better see how the course material is interconnected.
3. Sharing and discussing notes to future students encouraged me to reflect on my growth during this course.
4. The discussion increased my belief in my ability to execute the behaviors necessary to be successful in my college career.
5. The discussion helped me make meaning out of the overall theme of all topics that were covered.
6. The discussion helped me see the arc of the course—and how I've changed along the way.
7. The discussion made me feel that I had a voice in our class and gave me a sense of community and closure.

8. In what ways has this activity affected your perception of the course compared to what is normally done on the last day of college courses?

For questions #1 through #7, each statement was followed by:
[ ] Strongly Agree [ ] Agree [ ] Neutral [ ] Disagree [ ] Strongly Disagree
A "Comments/examples" piece was included following questions #1 to #7.

5 Results

The class sizes were small, but attendance was good as students were aware that an assigned activity would take place during class. We present the survey results and include a sample of student comments.

5.1. RETRIEVAL+REFLECTION: The course summary discussion helped to strengthen my recall and understanding of what was learned during the semester.

Table 1: Student Survey Responses

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Total	41	47	15	0	0
Percent	40%	46%	15%	0%	0%

All seven Likert-scale questions produced nearly identical results, so only the first is shown. These results indicated a strongly positive response from students. For each of the seven strategies evaluated, 77–86% of students selected either "Agree" or "Strongly Agree," demonstrating broad perceived usefulness. Notably, no respondents selected "Strongly Disagree," and the "Disagree" option was chosen by only 0–2% of students for all but one item.

- Thinking over what we have done over the entire course and having to decide which aspects were most difficult, impactful, surprising, etc. definitely helped in recalling the material. The discussion helped with understanding from hearing other students' opinions and thoughts on subject material.

- The course summary discussion helped me reflect on everything I have learned from the semester, specifically from the first day of class to now. I can see the growth in knowledge and understanding compared to then.

5.2. METACOGNITION AWARENESS: The course summary discussion helped me to better see how the course material is interconnected.

- I made some of my own connections, but it is also beneficial to hear from others. It gives a different perspective about concepts.

- The discussion in particular helped me think about how this course not only connects to itself, but also every other course the CS dept offers.

5.3. TEACHING & ELABORATION: Sharing and discussing notes to future students encouraged me to reflect on my growth during this course.

- It helped me to see what I would have done differently myself if I were to start over. I will take my own advice into my next class.
- I am proud of how I have done in this class, and it is important to recognize your own growth.

5.4. GROWTH MINDSET: The discussion increased my belief in my ability to execute the behaviors necessary to be successful in my college career.

- I really feel more confident about the class overall, and how each step of the way taught me something to be used in the future.
- So, looking back at what I've learned reminded me that I also can learn even the most difficult way; it's just persistence and curiosity. Now, this has given me more security in future computer science courses.

5.5. SCHEMA RECONSTRUCTION: The discussion helped me make meaning out of the overall theme of all topics that were covered.

- I was able to see how important everything was. I knew that all the topics would build from each other so having a strong foundation was important but seeing how they were fundamental to computer science helped put things in perspective.
- It helped me to see how the topics discussed not only connect to other topics taught in the course, but also to topics taught in other courses.

5.6. STUDENT VOICE: The discussion helped me see the arc of the course—and how I've changed along the way.

- Comparing how overwhelming this class felt when we first started to now is like night and day.
- The things that didn't make sense coming right after another totally do now that I can see the true theme of the course.

5.7. INTELLECTUAL CLOSURE: The discussion made me feel that I had a voice in our class and gave me a sense of community and closure.

- I really liked how the discussion had us reflect on where we started in the class and how much we've grown, not just academically, but personally too. It made me proud, not only of myself but of my classmates as well. Computer science isn't an easy subject, especially if you come in with no coding experience. Even though I didn't get to talk to everyone, I felt like we all shared similar challenges and experiences that helped us grow as students. In the end, the discussion made closing out the course feel meaningful, leaving me with a positive memory of this academic chapter.

- The discussion really helped me feel like I had a voice in our class and gave me a sense of community and closure. I'm usually pretty quiet (shocking), especially around people I don't know, and I tend to see my classmates as way ahead of me academically, which makes it tough to reach out. But this conversation made me realize that a lot of us share similar struggles and ways of thinking. It made me feel more confident about connecting with my classmates. Plus, it was a nice way to wrap things up instead of the class just ending abruptly.

5.8. SUMMARY QUESTION: In what ways has this activity affected your perception of the course compared to what is normally done on the last day of college courses?

- This is the best last day of class I've ever had. I could evaluate myself and see what to improve.

- The discussion was a good chance to reflect back on the course with my classmates and felt like a celebration of the course and the work we have done together as a class.

- It makes me feel that the professor/university actually cares about my opinion and wants to maintain a helpful learning environment for us and future students. Typically, the last day of classes is just cram before the final, but this showed interest from the department that I don't normally see.

- It felt like closure. Usually I get caught up in so much work that the semester feels like it flies by. This felt like a true end and allowed a second to sit back before finals and reflect on everything done before diving into studying and preparing for the final exam.

- Usually, the last day is quite high speed and quite boring, but this activity allowed me to reflect on what I have actually learned, my progression, and what I still need to work on. Much more useful than just goodbye as it gives a real relevance to all the hard work put into it.

6 Conclusions and Future Work

There were many benefits to the participating students. When looking at survey results and comments, students: (1) gained a sense of closure and satisfaction as the course came to an end; (2) recognized meaningful connections within the course content and to other CS classes; (3) took pride in the personal growth and development they experienced throughout the term; (4) felt reassured and more confident after learning that their peers faced similar challenges; (5) were surprised by how much they had actually learned during the course; (6) picked up useful strategies from classmates to apply in future CS courses; and (7) felt valued, knowing their professor cared enough to listen to their feedback.

By allowing time for individual reflection before discussion, the activity appeared to increase students' confidence when sharing their conclusions. Notably, several students who had been quiet throughout the semester participated confidently, which may explain comments expressing surprise at shared experiences within the cohort.

Not all 103 students felt as positive and enthusiastic as the majority. We liken it to receiving a critical comment in an otherwise positive set of course evaluations. While we thoughtfully consider all feedback, it is worth noting that the vast majority of the more than 800 comments were overwhelmingly positive and supportive. It is also worth considering some of these students may simply already have strengths in the area of metacognition and the activity did not necessarily make a change if they already held some of the characteristics.

Upon completing this end-of-semester activity, all three authors independently recognized it as an ideal way to conclude future computer science courses. We consider whether providing time for written reflection prior to the optional open discussion was an integral factor in fostering a stronger sense of connection within the cohort. Discussions are underway to refine and adapt the process for improved implementation in subsequent semesters.

We found the activity highly beneficial for computing courses, with strong potential for broader adoption across the college. To expand its reach, the authors presented the work in a Fall 2025 Teaching Talk, engaging faculty in discussion and offering support for adapting the activity to other courses. As adoption increases, we plan to systematically collect cross-disciplinary data to compare impacts across majors and refine the activity's structure and assessment, potentially focusing on a targeted subset of the seven strategies.

Our team envisions a possible study in which we see how the different Myers-Briggs personality types are impacted by the internal reflection and sharing exercise on the last day of class. Finally, our team realized that this process could be leveraged by applying these types of reflection and sharing activities earlier in the semester to create a sense of unity, belonging and collaboration

that would benefit student learning sooner in the course.

References

[1] Albert Bandura. *Self-Efficacy: The Exercise of Control*. New York, NY: W. H. Freeman and Company, 1997.

[2] Albert Bandura. *Social Foundations of Thought and Action: A Social Cognitive Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[3] Jane Chandler, Janet Carter, and Ian Benest. "Extravert or Introvert? The Real Personalities of Computing Students". In: *Proceedings of the 4th Annual LTSN-ICS Conference*. 2003.

[4] Jeffrey D. Karpicke and Janell R. Blunt. "Retrieval Practice Produces More Learning Than Elaborative Studying with Concept Mapping". In: *Science* 331.6018 (2011), pp. 772–775.

[5] Myers & Briggs Foundation. *The Preferences: E–I, S–N, T–F, J–P*. Accessed: May 30, 2025. 2025. URL: `https://www.myersbriggs.org/my-mbti-personality-type/the-mbti-preferences`.

[6] Garrett O'Day and Jeffrey D. Karpicke. "Comparing and Combining Retrieval Practice and Concept Mapping". In: *Journal of Educational Psychology* 113.5 (2021), pp. 986–1001.

[7] David S. Yeager and Carol S. Dweck. "What Can Be Learned from Growth Mindset Controversies?" In: *American Psychologist* 75.9 (2020), pp. 1269–1284.

[8] Todd Zakrajsek. *Don't Just Fade to Black: Ending a Course With Purpose*. The Scholarly Teacher. Apr. 2025. URL: `https://www.scholarlyteacher.com/post/don-t-just-fade-to-black-ending-a-course-with-purpose`.

[9] Todd Zakrajsek. *Essentials of the New Science of Learning: The Power of Learning in Harmony with Your Brain*. New York, NY: Routledge, 2025.

[10] Todd Zakrajsek. *The New Science of Learning: How to Learn in Harmony with Your Brain*. 3rd ed. New York, NY: Routledge, 2022.

Over a Decade of Computer Science Major Field Test Outcomes: Trends, Insights, and Perspectives*

Aziz Fellah
Computer Science and Information Sciences
Northwest Missouri State University
Maryville, MO 64468
afellah@nwmissouri.edu

Abstract

Administered across diverse U.S. institutions, the Major Field Test (MFT) in Computer Science benchmarks student knowledge in core areas and guides curriculum enhancement. It covers foundational topics—Programming, Software Engineering, Discrete Structures, Algorithms, and Systems—and includes a lightly weighted component on emerging areas such as Artificial Intelligence, Machine Learning, and Data Science. This paper presents a more than decade-long analysis of MFT results, examining trends in student performance and the impact of curriculum design, academic and transfer trajectories, student preparation, and course sequencing. Properly interpreting MFT scores within the curricular context, particularly considering the Standard Error of Measurement (SEM), is critical, as inconsistencies in course sequencing and student preparation may undermine comparisons with regional and national benchmarks. When applied as part of a program evaluation strategy, the MFT provides valuable insights for identifying curricular strengths and addressing learning gaps. This study presents evidence-based, strategic recommendations to guide curriculum development, improve student learning outcomes, and strengthen program assessment practices, supporting effective program evaluation.

1 Introduction and Related Work

This paper reflects on my experience as the Major Field Test (MFT) coordinator for more than a decade, highlighting insights, trends, and lessons learned from preparing Computer Science students for their final comprehensive assessment before graduation. Developed by Educational Testing Service (ETS) [6], the MFT is a widely used standardized test that evaluates student knowledge across core academic fields, with a focus on Computer Science. Many institutions use the MFT to assess student outcomes and align curricula with regional and national benchmarks [14, 16].

Within Computer Science, the MFT assesses fundamental areas such as Programming, Software Engineering, Algorithms, Discrete Structures, and Systems, and includes a lightly weighted component covering emerging topics such as Data Science, Artificial Intelligence, and Machine Learning. In its standardized form, the MFT provides benchmark data that guides curriculum development and program evaluation. Prior studies underscore its value in identifying curricular strengths and weaknesses. Zhang and Zhuang [18] showed how multi-year MFT data can identify curricular gaps and guide improvements that enhance student outcomes, while Gorka et al. [9] used the MFT in a self-study leading to course redesigns.

Most research on the MFT focuses on improving the assessment through question design, curricular alignment, or leveraging results to strengthen Computer Science programs. Abbott [1] examined the influence of explicit test preparation on student outcomes. MFT preparation activities are embedded within coursework or through activities outside the classroom, allowing students to become familiar with sample questions and test-taking strategies. Other studies have analyzed MFT performance across disciplines: Contreras et al. [4] examined business majors over time; Iqbal [11] explored curriculum and teaching improvements; and Mingxian [13] integrated the MFT into a Computer Science capstone to reinforce core concepts.

Additional research highlights the MFT's role in broader program evaluation. For example, Kirkland and Zhao [12] showed how MFT results reveal strengths and gaps in Science, Technology, Engineering, and Mathematics (STEM) programs, while Hunt and Sloan [10] discussed its usefulness for cross-program comparisons. Several authors described how MFT results influenced course redesigns [1, 2, 5], and Brock [3] used it to evaluate STEM program outcomes. Nursah [17] explored presents a systematic review of methods used to assess student competencies in computing education, highlighting trends and gaps.

Using over a decade of MFT data, this study explores how the MFT has guided ongoing academic program improvement. The MFT is widely recognized as a reliable tool for assessing student learning, curriculum effectiveness,

and guiding syllabus revisions, yet it is important to acknowledge both its benefits and limitations. Institutional policies shape how the MFT is administered: at some institutions, it is a graduation requirement, while at others it is not. Often, the MFT is embedded within coursework and contributes a modest percentage of the course grade to encourage student engagement. Students may be required to take the MFT, but individual scores do not affect graduation eligibility. This analysis also considers the student experience and evaluates the practical impact of the MFT on enhancing learning outcomes.

A central responsibility in this work has been translating MFT results into actionable curricular revisions. In collaboration with faculty, MFT domain-level outcomes were aligned with course-level learning objectives. Persistent underperformance in specific areas prompted targeted curriculum enhancements, including strengthened laboratory components and restructured assignments, which coincided with measurable increases in MFT mean scores over a four-year period. Although not required for accreditation, the MFT has become an influential component within program assessment and accreditation documentation.

The remainder of this paper is organized as follows. Section 2 presents the methodology and MFT topics, including a detailed Computer Science breakdown. Section 3 covers variables and analysis, including student preparation, transfer trajectories, subject-area performance, Standard Error of Measurement (SEM), and limitations. Section 4 reports results, highlighting patterns in core concepts. Section 5 discusses program insights, lessons learned, and recommendations for curriculum alignment. Section 6 concludes with key findings and implications for future MFT evaluation.

2 Methodology and Data Collection

This study analyzes over a decade of MFT results from a Computer Science program [7, 8, 15], including overall and sub-scores, student demographics (ethnicity, gender, educational level, transfer and enrollment status, language), and course sequencing. Institutional analyses were combined with MFT outcomes to trace students' academic trajectories and assess how course timing and prior preparation influence performance.

The MFT has served as a standardized assessment of students' mastery of core disciplinary concepts. Composed of multiple-choice questions across essential topic areas, it provides institutions with evidence of learning outcomes and supports both improvement and reporting. Although not always required for accreditation, MFT results are widely accepted as indicators of student achievement and program quality. This study focuses on the Computer Science version of the exam.

2.1 Data Sources and MFT Topics

The study includes more than a decade of MFT results and the dataset, encompassing total scores, sub-scores by area, and demographics [8, 15], as well as transfer status and indicators of course sequencing. Building on these data, the study analyzes institutional records to trace academic trajectories and examine how variations in timing, preparation, and progression influence performance, supporting the investigation of trends and curriculum alignment over time. For this study, we focus on those relevant to Computer Science students while providing context within the broader academic landscape.

2.2 Computer Science MFT Topic Breakdown

The Computer Science MFT covers several key areas, each weighted according to relative importance. Based on available ETS documentation [8, 15], the approximate weighting is estimated as follows:

- **Programming and Software Engineering**
  Core programming and software engineering principles, with related topics also included.
- **Discrete Structures and Algorithms**
  Core concepts in discrete mathematics and algorithm design.
- **Architecture, Operating Systems,Computer Networks and Databases**
  Core systems topics covers Computer Architecture, Operating Systems, Computer Networks, and Databases; additional related concepts may also be included.
- **General Topics**
  Emerging areas such as Artificial Intelligence (AI), Human–Computer Interaction (HCI), Machine Learning (ML), Information Systems (IS), and Data Science (DS), are lightly weighted on the exam.

Figure 1 summarizes the approximate relative weighting of each MFT subarea within the Computer Science major.

3 Variables and Analytical Approach

MFT outcomes were examined along several dimensions. Data were analyzed descriptively and comparatively, with visualizations and analysis of score distributions, performance trajectories, and subject sub-scores across student groups.

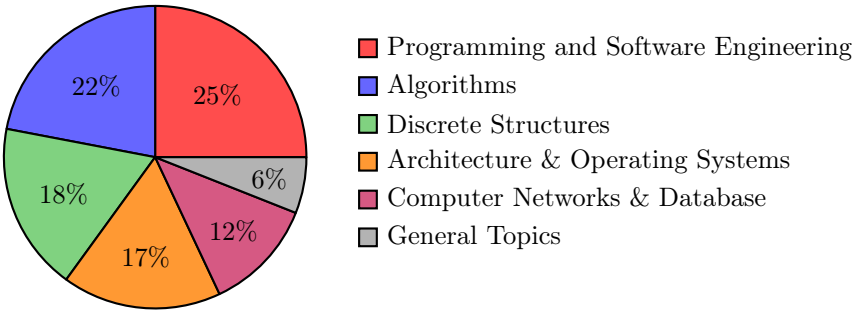

Figure 1: MFT Topic Breakdown for the Computer Science Major

- **Student preparation and curriculum alignment:** Results show that course sequencing, preparation, and completion of core courses strongly influence MFT outcomes. Students who took the test before completing key coursework—such as Programming Languages Concepts, Algorithms Design and Analysis, Programming, Software Engineering Principles, Discrete Structures, Databases, Computer Organization and Architecture, Computer Networks, and Operating Systems—often scored lower due to incomplete preparation. Proper alignment between course completion and test timing significantly impacted performance.
- **Transfer and course trajectories:** Differences between continuing and transfer students are examined, showing how varying course trajectories and transfer credits affect performance and overall test preparation. Curricular misalignment for transfer students has a noticeable impact on outcomes.
- **Subject-area performance:** MFT sub-scores across core areas—such as Programming, Software Engineering, Discrete Structures, Algorithms, and Systems—are analyzed to identify persistent strengths and areas of concern. Lightly weighted emerging areas, including Artificial Intelligence, Machine Learning, and Data Science, are also examined, though they have limited influence on overall score variability.
- **Standard Error of Measurement (SEM) and score variability:** Interpretation of MFT scores accounts for the Standard Error of Measurement (SEM), recognizing that scaled scores are estimates that may slightly under- or over-represent actual knowledge. Score distributions and variability are used to compare cohorts and interpret trends relative to institutional, regional, and national benchmarks.

- **Limitations:** This analysis is constrained by the level of detail available in ETS reporting, variability in student preparation, and statistical limitations associated with smaller sample sizes. These factors are considered when interpreting institutional, regional, and national benchmark comparisons.

4 Results and Analysis

Analysis of MFT outcomes revealed patterns associated with student preparation, course completion, transfer trajectories, and subject-area performance. For instance, students who had not completed key coursework in systems, programming, and algorithms tended to score lower, highlighting the importance of aligning test timing with course completion.

Differences between continuing and transfer students were noticed, with transfer students more likely to experience curricular misalignment affecting performance in core concepts. MFT scores in Programming, Software Engineering, Discrete Structures, Algorithms, and Systems reflected strengths when core courses were completed and weaknesses when coursework was incomplete. Lightly weighted emerging areas contributed minimally to overall score variability.

Score trajectories reflected the effects of curricular enhancements, including reinforced labs, revised assignments, and adjusted course sequencing, with improved performance following these changes. Overall, results indicate that student preparation and curriculum structure strongly influence MFT performance, underscoring the importance of carefully interpreting scores in the context of SEM, course completion, and student trajectories. Between September 2015 and June 2025, 221 domestic institutions across 38 states administered the test [15] (see Tables 1 and 2).

Assessment Indicator	**Institutions**	**Mean**	**Median**	**Std. Dev.**
Programming and Software Engineering	221	48.6	50.0	10.5
Discrete Structures and Algorithms	221	40.1	40.0	9.5
Systems: Architecture, Operating Systems Computer Networks, Databases	221	39.7	40.0	7.6

Table 1: Assessment Indicator Summary Statistics.

Number of Examinees	Number of Institutions	Mean	Median	Std. Dev.
28,271	221	147.6	147.0	15.4

Table 2: 2025 Comparative Data Guide - MFT for Computer Science Total Score Distribution.

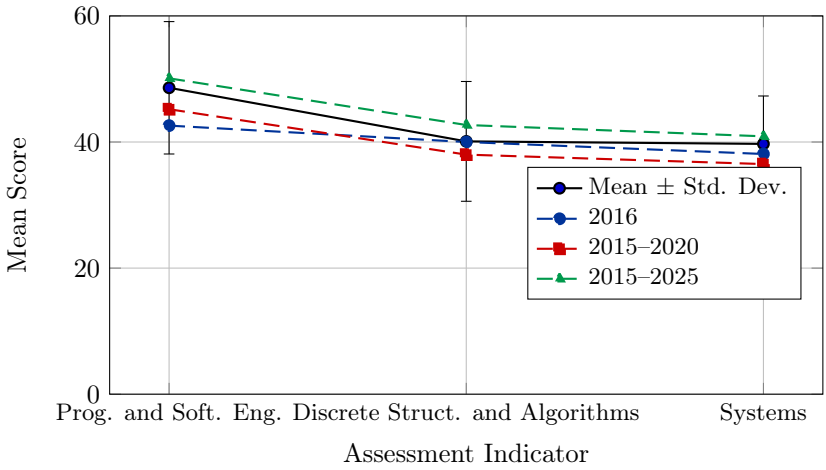

Figure 2: Comparison of Computer Science MFT Assessment Indicators Across Multiple Time Periods.

4.1 MFT Performance Across Core Areas

Figure 2 shows student performance in four core Computer Science MFT areas over time, comparing institutional and regional mean scores.

- **Programming and Software Engineering:** Institutional mean aligns with the regional mean, reflecting consistent instruction in these fundamental areas.
- **Discrete Structures and Algorithms:** Institutional mean lags behind the regional mean, highlighting areas for curriculum enhancement, particularly in algorithm design and analysis.
- **Systems: Computer Architecture, Operating Systems, Databases, Computer Networks, Databases:** Institutional mean exceeds the regional mean, showing competency in core systems topics, with room to strengthen integrated system concepts.
- **General Topics:** Small differences between institutional and regional

means indicate broad understanding across foundational areas.

5 Discussion

Analysis of more than ten years of MFT data highlights how curriculum sequencing, test timing, transfer trajectories, and student preparedness shape performance in core Computer Science domains. Alignment across regional, national, and institutional results suggests that the curriculum effectively prepares students, though gaps—particularly in Discrete Structures and Algorithms—reflect course timing and incomplete preparation.

Differences between continuing and transfer students illustrate how varied trajectories and prerequisites influence performance in Algorithms, Discrete Structures, and Systems. These findings underscore the importance of interpreting MFT data within the context of academic trajectories and the Standard Error of Measurement (SEM), rather than as absolute measures of learning.

Curricular enhancements, including strengthened labs, revised assignments, and improved sequencing, led to measurable gains in sub-scores, demonstrating the value of refinement. Overall, results emphasize evaluating MFT outcomes at the program level, where aggregated data provide meaningful benchmarks for long-term instructional effectiveness.

5.1 Interpreting MFT Scores and Program Insights

Understanding the MFT scoring system is essential for accurate interpretation. Total scores range from approximately 120 to 200 and represent scaled, not raw, results; low scores reflect statistical normalization rather than low mastery. ETS Assessment Indicators [7], expressed as percentages correct by subject area, allow comparison with regional and national benchmarks and identification of strengths and weaknesses. Interpreting scores also requires considering SEM, academic trajectories, and curricular timing, particularly when students take the exam before completing core coursework. Multi-year trends provide more meaningful insights than single-year results, making the MFT most valuable as a program-level assessment tool

Multi-year data highlight key lessons, indicating that aligning course sequencing with MFT content, improving faculty-advisor communication, and supporting transfer students are critical for preparation and performance. Curricular enhancements—including reinforced labs, revised assignments, and adjusted sequencing—led to measurable gains. Recommendations include offering the MFT in both semesters, reinforcing alignment with instruction, supporting varied student trajectories, interpreting results in the curricular context, and using the MFT for program-level assessment rather than individual evaluation.

6 Conclusion

Over a decade-long analysis of Computer Science MFT results demonstrates the value of integrating standardized assessment into curriculum improvement. Performance in Programming, Software Engineering, Discrete Structures, Algorithms, and Systems reflects course sequencing, academic trajectories, and student preparation. Curricular enhancements contributed to improved subscores and closer alignment with regional and national benchmarks. Over ten years, the MFT was administered at 221 U.S. institutions, with more than 28,000 students participating. Emerging topics such as AI, Machine Learning, and Data Science are not included in this study, and most institutions administer the MFT as an exit assessment rather than a graduation requirement. When applied thoughtfully, the MFT can strengthen long-term academic quality and support program-level evaluation in Computer Science.

Acknowledgments

The author thanks the Assessment Office at Northwest Missouri State University for support with the MFT, and Educational Testing Service for providing valuable input on the Computer Science Major Field Test.

References

[1] Russ Abbott. "An Assessment of the Computer Science Major Field Achievement Test (MFAT)". In: (2010).

[2] Richard G. Baldwin. "Assessing Business Program Effectiveness Using MFT". In: *Business Education Forum* 64.3 (2010), pp. 12–17.

[3] William A. Brock. "Using the MFT to Evaluate STEM Program Outcomes". In: *Journal of STEM Education* 4.2 (2023), pp. 12–20.

[4] Salvador Contreras et al. "Documenting and explaining major field test results among undergraduate students". In: *Journal of Education for Business* 86.2 (2011), pp. 64–70.

[5] Steven Davis and Thomas Meyer. "Integrating the Major Field Test in Computer Science Capstone Courses". In: *Journal of Computing Sciences in Colleges* (2015).

[6] Educational Testing Service. *Computer Science Major Field Test: Test Description and Content Validity*. Tech. rep. ETS, 2021.

[7] Educational Testing Service. *Guide to Score Interpretation and Validity Studies for Major Field Tests*. Tech. rep. ETS, 2024.

[8] Educational Testing Service. *Internal Report on MFT in Computer Science*. Unpublished internal document. Interested readers may contact the author for further information. 2025.

[9] Steven Gorka, Jim Miller, and Dhundy Shrestha. "A Self-Study Using the ETS Major Field Test in Computer Science". In: *Information Systems Education Journal* 8.42 (2010), pp. 1–13.

[10] Lisa M. Hunt and Robert Sloan. "Major Field Tests: Benchmarking Student Learning Across Disciplines". In: *Assessment Update* 26.6 (2014), pp. 5–7.

[11] Zahid Iqbal. "An Analysis of the Educational Testing Service Major Field Test for Business Performance: Further Evidence". In: *Journal of Educational Research and Practice* 10.1 (2020), pp. 242–254.

[12] David Kirkland and Li Zhao. "Using Major Field Test Results for Continuous Improvement in STEM Programs". In: *International Journal of STEM Education* (2019).

[13] Jin Mingxian. "Redesign of the Computer Science Capstone Course by Integrating the Major Field Test (MFT)". In: *Journal of Computing Sciences in Colleges* 24.1 (2018), pp. 239–246.

[14] Educational Testing Service. *About the Major Field Tests*. Accessed: 2025-11-26. 2025. URL: `https://www.ets.org/mft/about.html`.

[15] Educational Testing Service. *Correspondence Regarding Computer Science Major Field Test content and Weighting*. Educational Testing Service. 2025.

[16] Educational Testing Service. *Major Field Tests: Annual Comparative Data Guide*. 2025. URL: `https://www.ets.org/mft`.

[17] Nursah Yakut. "Systematic Review of Competency Assessment Methods in Computing Education". In: *Proc. of the 56th ACM Technical Symposium on Computer Science Education (SIGCSE TS 2025)*. ACM, 2025.

[18] Lei Zhang and Yu Zhuang. "Curriculum Assessment Using the Major Field Test in Computer Science: A Multi-year Study". In: *Journal of Computing Sciences in Colleges* 34.5 (2019), pp. 36–42.

A Light-Weight Approach to Standards-Based Grading for New Adopters*

Melissa Lynn
Department of Mathematics, Statistics, and Computer Science
St. Olaf College
Northfield, MN 55057
lynn5@stolaf.edu

Abstract

We describe a straight-forward process for converting exams in computer science courses from traditional grading to standards-based grading, with the goal of improving student learning and better aligning assessments with intended learning outcomes. While other approaches to standards-based grading can intimidate new adopters with complicated gradebooks and extensive student revisions, this approach is particularly tractable for instructors without prior alternative grading experience. The process of adopting this approach is described in detail, with practical recommendations. We found that changing to standards-based grading positively affected students' demonstrated learning and experience in their courses. In addition, instructors had positive experiences that encouraged them to further explore alternative grading methods.

1 Background and Related Work

As defined by Clark and Talbert [2], *traditional grading* is characterized by a lack of opportunity for reattempts or revisions of student work, assignment of a point or percentage value, and a letter grade determined by a weighted

average of all work submitted during the semester. In contrast, *alternative grading practices* incorporate feedback loops and more directly connect grades to demonstrated learning outcomes. One approach to alternative grading is *standards-based grading*, on which this paper will focus. In standards-based grading, the instructor develops a list of *standards* or learning outcomes, describing what a student will learn to do in the course. Assignments are developed around these standards, and each standard is assessed multiple times without a penalty to the student's grade. Students are graded on whether they have met the standards by the end of the semester. *Standards-based testing* refers to standards-based grading applied specifically to testing.

There is a growing body of work demonstrating the efficacy of alternative grading in computer science. Chen et al. report faster, more consistent grading and clearer expectations when using standards-based grading [1]. Tuson and Hickey found that mastery learning and specifications grading resulted in strong student mastery in skill areas [8]. Weber found that alternative grading was effective for promoting learning for non-CS majors [9]. Lionelle et al. added revisions with positive impacts on student success [5]. Spurlock reports increased student effort and perceived learning in electives evaluated with ungrading [7]. Mattfeld measured improvement in student motivation when using alternative grading [6]. Fine found that an alternative grading scheme resulted in a deeper understanding of material [4]. In Deeb et al., students reported reduced stress and more complete learning [3].

While instructors may want to explore alternative grading, there can be significant obstacles to adopting a new grading practice. Instructors may be concerned about the time and effort required. If students, colleagues, or their institution are not receptive to a new approach to grading, this could impact job security. Scalability for large courses could be a concern, as well as gradebook management for nontraditional grades.

In this paper, we describe an approach to standards-based testing that seeks to mitigate many of these obstacles, providing an on-ramp for instructors who are interested in exploring alternative grading, but may be hesitant about the difficulty of a transition. This approach replaces traditionally graded in-class exams with in-class exams using standards-based grading, without affecting the other assignments in the course. This allows the instructor to make an incremental change without redesigning an entire course. Student reception to this approach has been very positive, as has instructor feedback.

2 Developing Standards

To develop standards for a course, we follow the process of backwards design, which centers course development on intended learning outcomes. Wiggins

and McTighe outline three stages of backwards design, which we translate into developing standards for standards-based testing [10].

Identify desired results. This stage involves deciding: what should a student understand and be able to do at the end of the course? At this stage, we begin to develop the descriptions of the standards, determining the most important skills that a student should have developed by taking the course. The instructor should consider what they think is most important for students to take out of the course, as well as what has been taught previously, and the expectations of students and colleagues. The context of the course as a prerequisite, in a computer science major, or in general education requirements may also be considered. They may account for the skills that are most useful for students as they enter the workforce or in the long term. At this stage, we prioritize the most *essential* outcomes for the course. We have found that between 10 and 15 standards work well for the approach described in this paper. Note that even if a skill or topic is not part of a course standard, it can still be incorporated into the course through classwork or homework.

Determine acceptable evidence. At this stage, we clarify how we will assess the standards of the course, asking: how will a student demonstrate that they have achieved this goal? Some standards may be well-suited to assessment through testing, while others may be better assessed through projects or other means. With that in mind, we write our standards as specific, assessable learning outcomes. These are statements of the form "By the end of this course, students will be able to...," completing the sentence with an *assessable* skill that students should have achieved. By *assessable*, we mean that it must be possible for the student to demonstrate that they have achieved this outcome, and the instructor must be able to judge whether the student has achieved the outcome.

Plan learning experiences and instruction. It is important to clearly communicate the standards and expectations for passing the standards, and to provide opportunities for students to practice for the assessments. We dedicate class time to explaining each of the standards and what we will specifically be looking for to see that a student has demonstrated proficiency in the standard. We also post a practice packet for each standard, including the explanation and expectations for the standard, along with many practice exercises. We have required students to complete practice quizzes and exams. This is in addition to class and homework time spent introducing and practicing the skills related to the standard.

Examples of standards developed for various computer science courses, as well as an example practice packet, are available at `https://github.com/melissa-lynn/sbt_paper`.

3 Quiz and Exam Structure

Each standard is graded on the following scale. Awarding no credit (I), half credit (S), or full credit (P) for each standard enables entering scores in a traditional gradebook. In our courses, the course standards are typically worth 35-45% of a student's grade.

Proficient (P) indicates that the student demonstrates proficiency in the standard, and they are awarded full credit. The expectations for a P are very high; essentially all problems for the standard must be solved completely correctly. The only errors that can still earn a P are very minor errors unrelated to the standard, such as a small arithmetic error.

Revision (R) indicates that the student's work demonstrates proficiency in the standard, but has some minor issue that they need to address before earning a P. This includes errors such as missing an answer for part of one question, skipping some detail in work shown, or a mistake unrelated to the standard that we should discuss. These kinds of errors would typically be "-1 errors" in traditional grading. When deciding between a P and an R, we consider whether we want to be sure that the student corrects the error.

To complete a revision, students write up a revised solution with an explanation of their changes, and bring this to office hours to discuss. These revisions are open-resource. If the student does not complete the revision, or the revision is unsuccessful, the grade is changed to an S.

Starting (S) indicates that the student's work shows partial proficiency in the standard, and they earn half credit. They have the opportunity to reattempt the standard on a future exam. An S is typically awarded when the student successfully completes half or more of the questions on the standard, but at least one is answered substantially incorrectly. To decide between awarding an S and an R, we consider the question, "based on this work, do I think this student would benefit from studying the standard again?" If the student receives an R, they are able to revise their existing solutions on the same problems, with open resources. If the student receives an S, they must reattempt the standard, with a new set of questions, on the next closed-resource exam.

Incomplete (I): indicates that the student's work does not yet demonstrate proficiency in the standard, and earns no credit. They are able to reattempt the standard on a future exam. An I is typically awarded when the student does not attempt the questions corresponding to the standard or answers the majority of the questions incorrectly. To decide between awarding an I and an S, we consider whether the student has made progress towards proficiency in the standard.

Students have three attempts to earn a P on each standard, all on in-class closed-resource assessments. The first attempt on each standard is on a quiz,

the second on an exam during the semester, and the third and final attempt is on the final exam. Each quiz is given for half of a class period (about 25 minutes), and typically covers two standards. There are two exams given during the semester for a whole class period (55 minutes), with the first exam around the midpoint of the semester, and the second close to the end of the semester. Each exam covers half of the standards from the course. The final exam covers all standards from the course.

Once a student earns a P on a standard, they receive full credit for the standard, and do not need to repeat it on future exams. Only standards that the student has not yet completed appear on subsequent exams. This means that if a student receives a P on all standards on quizzes and earlier exams, there is no final exam for that student to take. If a student receives a lower grade on a standard than they had on a previous assessment (for example, if they received an S on a quiz but an I on the final exam), they retain the higher grade. We adopted this policy to reduce stress for students on the reattempts, so that their grades could only improve. Since different students have different subsets of standards to attempt on their exams, we use a Python script to generate a custom exam for each student with their remaining standards.

We have used the approach described in this paper for classes with as many as 60 students, though it could be adapted to larger class sizes. Instead of meeting in office hours, students can write a revised solution with an explanation of their changes and submit these to a TA for regrading. We used this modification in a CS 1 class with 60 students, and this approach to revisions took a minimal amount of time. For distributing custom exams in a large class, an alphabetized seating chart can be used so that exams can be presorted to be passed out quickly. We have found grading the standards-based exams to be a bit quicker than grading traditional exams, because the simplified grading scale results in less time deciding partial credit amounts. As a result, we expect that the approach in this paper will be scalable to large classes.

4 Results

To evaluate the effectiveness of this approach, we use the author's observations as an instructor, student feedback, and a comparison between exam results in a course given with traditional grading and with standards based grading. We also include reports on the experiences of three additional instructors who have used this approach to standards-based testing.

4.1 Exam Comparison

We compare the proficiency demonstrated by students in the same course in two consecutive semesters, one with traditional exam grading and one with

Table 1: Comparing Exam Structures

Standard	Exam Structure	P/R	S	I	p (P/R)
DR	Traditional	52.9%	35.3%	11.8%	0.0003
	Standards-Based	96.3%	3.7%	0.0%	
LR	Traditional	64.7%	23.5%	11.8%	0.1119
	Standards-Based	81.5%	18.5%	0.0%	
AP	Traditional	68.8%	12.5%	18.8%	0.0010
	Standards-Based	92.6%	3.7%	3.7%	
HC	Traditional	37.5%	56.3%	6.3%	0.0000
	Standards-Based	81.5%	14.8%	3.7%	
EP	Traditional	18.8%	25.0%	56.3%	0.0122
	Standards-Based	74.1%	22.2%	3.7%	
SF	Traditional	41.2%	41.2%	17.6%	0.0296
	Standards-Based	74.1%	22.2%	3.7%	

standards-based testing. The course considered serves as an introduction to computer systems, computer organization, and computer architecture. From the Fall 2023 final exam (traditional grading), we selected problems that assessed the Spring 2024 course standards.[1] For these problems, we regraded the Fall 2023 final exams with the criteria for the standards, producing P/R/S/I grades. Regrading the Fall 2023 exams allowed us to compare how many students achieved each level of proficiency in the standards in Fall 2023 (traditional grading) and in Spring 2024 (standards-based grading). Because Fall 2023 students had no opportunity for revisions, we grouped P's and R's together.

The results of these computations are included in Table 1. For example, for Data Representation (DR), 52.9% of students demonstrated proficiency on the final exam in Fall 2023 (traditional grading), compared with 96.3% in Spring 2024 (standards-based grading). For each standard, the percentage of students demonstrating proficiency was much greater when the course was taught with standards-based grading than with traditional grading. This difference was statistically significant ($\alpha = 0.05$) for all standards except Logic Representation (LR), with p values in the column labeled $p(P/R)$.

This comparison is imperfect. The instruction in Spring 2024 may have been somewhat more effective, since we had more experience teaching the course. The comparison uses only the final exam from Fall 2023, while the data from Spring 2024 include quizzes and exams earlier in the semester. We recall that

[1]The standards selected were Data Representation (DR), Logic Representation (LR), Basic Assembly Programming (AP), Hardware Components (HC), Executing a Program (EP), and Stack Frames (SF).

exams earlier in the semester in Fall 2023 showed similar results to the final, so we do not believe this contributed significantly to the difference in outcomes across the two semesters. That said, we assert that the consistently large differences in proficiency rates provide evidence that the standards-based testing approach positively impacted student learning.

4.2 Student Feedback

In this subsection, we summarize student feedback from midsemester surveys and end of semester course evaluations from several courses that have used this approach to standards based testing, from Fall 2023 through Fall 2024.[2] These responses were very positive about standards-based testing, although they included frustrations with some aspects of the implementation. We include some common themes from the comments.

Students generally felt that standards-based testing made the course less stressful, due to the safety net of extra attempts. However, a couple of students found standards-based testing more stressful, because of the high expectations for earning a P on a standard. Students commented that it helped them learn the material more effectively. They noted that it supported their learning process, and it was helpful to be able to focus their efforts on areas of weakness. Students wrote that this approach should be retained when the course is offered in the future. Some students wrote that they found the grading too strict, and that the expectations for solutions were too rigid. They also commented that the distinctions between the different grades, especially S and R, were not always clear. Some students requested less all-or-nothing grading, suggesting that a 75% mark would be nice. Students also wrote that they would like to see solutions to practice problems and quizzes, so they could get a better idea of what was expected for proficiency.

From this feedback, we believe that the benefits to students outweigh the drawbacks, and that the frustrations can be mitigated with better communication of expectations for demonstrating proficiency. We do not plan to add a 75% mark, as we expect that this would reduce the incentive to earn a P, negatively impacting student learning.

4.3 Author's Observations

In courses adopting the standards-based testing approach from this paper, we have observed many positive effects. Students demonstrated better mastery of the material and were motivated to learn concepts completely. The approach

[2]The courses included are Analysis of Algorithms (54 responses), Theory of Computation (28 responses), and Hardware Design (22 responses).

enabled high expectations by allowing reattempts, though these high expectations were sometimes frustrating for students, as observed in Subsection 4.2. It was especially gratifying to work with students who struggled to pass standards and learn the material early in the semester, but worked hard to improve and were able to earn a P on the standards by the end of the semester. Grading was a bit quicker and more consistent, since there was no need to spend time deciding on partial credit. Because students needed to come to office hours to complete revisions, there was more one-on-one interaction with students. The end-of-semester exam scores in standards-based testing were much higher than those awarded in traditional grading, resulting in higher overall course grades. This may appear to be grade inflation, but it was correlated with better mastery of the course material demonstrated by students.

The most challenging aspect of converting to standards-based testing has been communicating the system and expectations to students. Even with written descriptions in the syllabus and multiple in-class explanations, students are usually unsure of the system until they have gone through at least one quiz. It is also essential to communicate clear expectations for demonstrating proficiency in a standard, which can be done by providing study guides. The demands on class time have also been a challenge, as scheduling quizzes and exams for multiple attempts required more class time than traditional exams.

4.4 Instructor Experiences

In this subsection, we provide feedback from three instructors at two liberal arts colleges who were introduced to this approach by the author and have used the approach to standards-based testing in their courses.

Instructor A used the approach in a 50 student introductory data science course. This instructor had prior experience with different forms of standards-based grading, but was new to this particular form of standards-based testing. They observed that this approach emphasized learning by encouraging students to revisit material they did not yet understand completely. They believe that the approach more accurately reflects students' knowledge than traditional grading. They noted that while it reduced stress for some students, some high performing students experienced increased stress because they put pressure on themselves to get everything correct the first time. They also observed that some students with significant deficiencies in understanding may previously have skated by with partial credit, but were unable to do so in standards-based testing. Instructor A found that grading was easier, because it took the "guess work" out of quantifying the level of understanding. However, they did sometimes find it difficult to decide on offering half-credit (S), rather than simply all or nothing. They noted that this approach still requires significant upfront time for planning, organizing the course schedule, and developing assessable

standards.

Instructor B has used this approach in CS 1 and Introduction to Machine Learning over multiple semesters, with between 20 and 30 students per course. This instructor had no prior experience with alternative grading, and has since explored additional approaches in their courses. Instructor B found that the approach was helpful to reassure students that failure is part of learning and to clearly articulate what they hoped students would learn. They found that students were less stressed by this approach to grading. Once they had written standards and study guides, preparing for each semester was straightforward. Instructor B did spend more time grading, but they also found that more checkpoints built into the semester made early intervention possible for struggling students. They found that the students' overall course grades increased significantly. Instructor B added the requirement that students' needed to pass 75% of the standards in order to pass the course, since they found that some students were earning a B- while only passing 30% of the standards.

Instructor C is currently using this approach for the first time, in CS 1 (28 students) and Analysis of Algorithms (15 students). They had no prior experience with alternative grading. Although they acknowledge that it is still early for them to judge, Instructor C thinks that the approach provides clear expectations and reduces stress, by allowing students to redo standards. As an instructor, they are focused on students achieving a clearly defined outcome, without an issue of a curve. Instructor C does wonder if content retention will suffer without a comprehensive final exam, and they wonder if the specificity of the objective could cause students to miss some aspects of a broader topic.

When asked if they would recommend this approach to someone who is new to alternative grading, all three instructors said that they would. Instructor B added the caveat that the instructor should have enough prior familiarity with the course to develop their standards. Instructor A noted that they think this approach is a great way to implement alternative grading without changing course assessment and structure completely. They would love to see anyone who adopts this approach explore more aspects of alternative grading, and they believe this is a great starting point.

5 Conclusion

The approach described in this paper provides an accessible on-ramp to standards-based testing for instructors who are interested in exploring alternative grading practices. We hope that instructors will be encouraged to try this approach, and that their experiences will lead them to explore and develop further alternative grading practices that support their students' learning experiences.

References

[1] Lijun Chen et al. "Experience Report: Standards-Based Grading at Scale in Algorithms". In: *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. ITiCSE '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 221–227. DOI: https://doi.org/10.1145/3502718.3524750.

[2] David Clark and Robert Talbert. *Grading for Growth: A Guide to Alternative Grading Practices that Promote Authentic Learning and Student Engagement in Higher Education*. New York, NY, USA: Routledge, 2023. DOI: https://doi.org/10.4324/9781003445043.

[3] Fatima Abu Deeb, Ella Tuson, and Timothy J Hickey. "Grading for Equity in a Hyflex Compiler Design Course". In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSETS 2025. New York, NY, USA: Association for Computing Machinery, 2025, pp. 4–10. DOI: https://doi.org/10.1145/3641554.3701835.

[4] Benjamin T. Fine. "Competency and Equity Driven Grading System for Computer Science Curriculum". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, pp. 311–317. DOI: https://doi.org/10.1145/3649217.3653564.

[5] Albert Lionelle et al. "A Flexible Formative/Summative Grading System for Large Courses". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 624–630. DOI: https://doi.org/10.1145/3545945.3569810.

[6] Ryan Stephen Mattfeld. "Improving Student Motivation through an Alternative Grading System". In: *J. Comput. Sci. Coll.* 39.5 (Nov. 2023), pp. 86–95. ISSN: 1937-4771.

[7] Scott Spurlock. "Improving Student Motivation by Ungrading". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 631–637. DOI: https://doi.org/10.1145/3545945.3569747.

[8] Ella Tuson and Tim Hickey. "Mastery Learning and Specs Grading in Discrete Math". In: *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. ITiCSE '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 19–25. DOI: https://doi.org/10.1145/3502718.3524766.

[9] 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. New York, NY, USA: Association for Computing Machinery, 2023, pp. 638–644. DOI: https://doi.org/10.1145/3545945.3569765.

[10] Grant P. Wiggins and Jay McTighe. *Understanding by Design*. 2nd ed. Pearson, 2005.

When Mentorship Isn't Enough: A Case Study in Undergraduate ML Research at an HSI*

Olivera Grujic
Computer Science
California State University Stanislaus
Turlock, CA
ogrujic@csustan.edu

Abstract

This paper presents a case study in mentoring undergraduate research in machine learning at a Hispanic-Serving Institution without a formal machine learning curriculum. A transfer student was guided through an effort to reproduce results from a public health study using real-world data on school-level tobacco-free policy implementation in India. While the student was supported through funded research programs and presented early findings at poster sessions, engagement waned as challenges arose in model performance, reproducibility, and data preprocessing. Despite technical outcomes being limited, the mentoring process surfaced key educational lessons in accountability, project scope, and the importance of scaffolding research skills in environments with uneven preparation. This work highlights the promise and pitfalls of undergraduate research in socially meaningful, data-driven computing at teaching-focused institutions.

1 Introduction

Involving undergraduates in meaningful research helps bridge the gap between classroom learning and applied computing. Undergraduate students lack the

structured infrastructure seen in production-oriented ML courses [5]. Students without coursework in probability or linear algebra can still learn ML effectively when given curated data and structured mentorship [6].

For students from underrepresented groups in computer science, mentored research experiences can be especially impactful in developing technical skills, confidence, and persistence as noted in prior work on student demographics and diversity at HSIs [4]. This paper presents a case study in mentoring an undergraduate student through a machine learning (ML) project using real-world public health data.

The project's goal was to reproduce results from a graduate-level study using machine learning models. The student participated through federally supported programs at a Hispanic-Serving Institution (HSI), which provided financial support and mentoring infrastructure. While the student showed strong interest in programming, limited prior preparation and inconsistent follow-through led to mixed technical outcomes. This finding aligns with recent observations that students often mistake functional code for research insight [7]. Nevertheless, the project revealed both the potential and challenges of mentoring undergraduate ML research at teaching-focused institutions.

The mentoring process emphasized the importance of scaffolding complex tasks, promoting academic ownership, and aligning technical work with socially meaningful problems. Prior work has shown that students without formal training in probability, linear algebra, or pipelines benefit from carefully curated datasets and instructor guidance when learning machine learning [6]. In parallel, studies at HSIs highlight how structured mentorship and culturally relevant projects can support retention and learning for underrepresented students in computing [4].

This paper describes the project and its outcomes, emphasizing both the technical lessons and mentoring insights relevant to educators in computing fields. While it focuses on the mentoring process and its educational implications, the student's technical task and dataset are presented to provide context, rather than to report empirical research results.

2 Institutional and Student Context

This project took place at a Hispanic-Serving Institution, where many students are the first in their families to attend college and often balance academic work with jobs or family obligations. The student who participated in this project was majoring in computer science and had a strong personal interest in programming and machine learning in particular. The institution does not currently offer a dedicated course in machine learning, and the computer science program is undergraduate-only. The student was a transfer student who joined

in his junior year.

He was selected through competitive application to two undergraduate research programs aimed at supporting underrepresented students in STEM fields: STEM CRU (Creating Research Opportunities for Undergraduates) and LSAMP (Louis Stokes Alliance for Minority Participation). These programs provided funding for the student to participate in research over the course of more than a year. The financial support was intended to free the student from taking on unrelated work so that he could focus on the research project. Early on, the student was enthusiastic, and his motivation showed through his willingness to present posters summarizing the early goals of the project. He presented three times in Spring 2024 and showed promise in learning new tools and understanding the dataset.

However, as the project progressed and it became clear that reproducing the graduate study's results would be difficult, the student's engagement began to drop. Despite repeated conversations about realistic outcomes and the nature of research, the student eventually chose to skip two scheduled presentations in Spring 2025. He updated his poster, but without any real progress, and chose not to attend the poster session. When encouraged to submit a research paper based on the project, he did submit a draft, but it was incomplete and unpolished, submitted in haste just before the deadline. The paper was ultimately rejected, and while the reviewer comments were fair and expected, the student responded by stepping away from the project entirely.

While the technical outcomes were limited, the experience reinforced the broader educational value of undergraduate research: navigating ambiguity, managing time, and learning to receive and act on constructive feedback. These are lessons that extend beyond machine learning or programming and they are sometimes learned the hard way.

3 Mentorship Model and Learning Goals

The mentorship model for this project was built around one-on-one, biweekly meetings during the academic semester. These meetings were scheduled to provide structure and accountability, with the goal of helping the student make steady progress on a challenging machine learning task over time. Although the student was officially funded to work 10-15 hours per week through undergraduate research programs, actual progress was often limited. Meetings frequently turned into check-ins with little or no new work completed between sessions.

During exam periods, campus breaks, and the summer (when funding restrictions applied), communication became inconsistent, and momentum was hard to sustain. In practice, the student appeared to treat meeting atten-

dance as the primary deliverable, rather than the research work itself. While understandable for an undergraduate still developing time management and self-directed work habits, this pattern became a barrier to deeper engagement with the project.

From the outset, the mentorship aimed to teach foundational concepts in machine learning and data science. Topics included logistic regression, support vector machines, neural networks, model evaluation, imbalanced data, and the interpretation of ROC curves. The student was also introduced to basic principles of writing for research, including how to write an abstract, structure a paper, and present research in poster format. When he encountered difficulties, they were often not technical dead ends, but rather points where effort had stalled. Several course corrections were needed when the student requested a new round of funding for continued work. At that point, he was asked to document and reflect on what had been done so far and submit a research paper. The goal was not to present novel results, but to learn how to communicate findings clearly and honestly. That paper was ultimately submitted and rejected, but it served as a valuable capstone and a teaching opportunity.

While direct writing assistance was not permitted under program guidelines, extensive support was provided in the form of outlines, figure suggestions, structural feedback, and targeted comments. Despite this, the student struggled to follow through. In one instance, he was asked to recreate a table from a paper he had read but submitted it verbatim without citation. He was then directed to replace it with a newer version from an updated source, but the revision was never completed. These moments illustrated an ongoing challenge: helping students understand the expectations of scholarly work and the importance of academic integrity.

In addition to technical instruction, the mentorship included support for developing essential research skills: how to read and interpret scientific papers, how to give a poster presentation, how to write a research abstract, and how to manage long-term projects. Time management was a recurring theme. The student was encouraged to break work into smaller tasks, work consistently rather than in bursts, and leave time for debugging and iteration, especially when working with unfamiliar machine learning tools.

This paper focuses on a particular student who showed early interest in ML and began developing models, but shifted to a different faculty project when funding discontinued, cutting off communication with the original mentor. However, this was not an isolated experience of mentoring students in the STEM CRU program. Three more students participated in weekly meetings for a semester or two and got paid full weekly hours. One of them made initial progress on supervised learning tasks but discontinued work immediately upon notification that he was no longer eligible for funding due to certain restrictions

pertaining to his case. Despite having access to equipment, peer support, and faculty mentorship, these students never produced written work or prototypes. Ultimately, the mentoring process reflected the broader reality of undergraduate research: technical guidance is essential, but so is helping students learn how to show up, stay focused, and respond constructively when things don't go as planned.

4 Research Task: Reproducing a Social Impact Study with Machine Learning

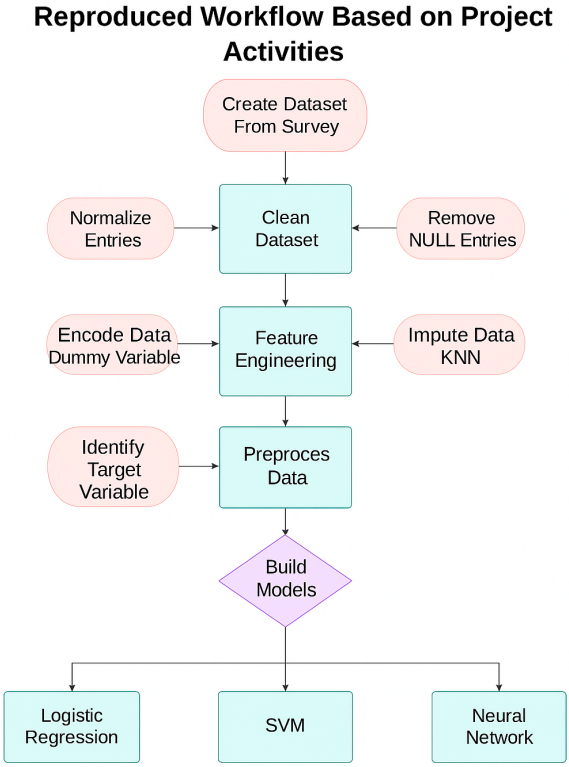

Figure 1: Reproduced Workflow Based on Project Activities

The student's technical task was to reproduce the results of a graduate-level study that used machine learning models to predict whether Indian schools would meet criteria for being designated "tobacco-free." The goal was to reinforce foundational machine learning skills: building models, selecting features, handling real-world data issues, and evaluating results using appropriate metrics. The student was introduced to the original study's modeling pipeline. Reproduced Workflow Based on Project Activities is shown in Figure 1.

Table 1: Description of the 11 Tobacco-Free School (TFS) Criteria and Percentage of Schools Meeting Each

No.	**Description of Tobacco-Free School Criteria**	**Percent of Schools (n=507)**
1	Posters in schools stating smoking in and around schools is not allowed	63% (318)
2	Posters stating the ill effects of tobacco and tobacco control laws inside the premises	45% (228)
3	Principal has a copy of directives/-circulars based on the 2003 law	52% (262)
4	Presence of a banner or poster near the entrance stating that this is a tobacco-free school	35% (175)
5	Tobacco selling is completely banned inside the premises and within 100 yards	74% (365)
6	No tobacco use inside the school	59% (298)
7	Tobacco control committee is in place and quarterly meetings are conducted	31% (157)
8	Tobacco control is part of the annual school health activities	39% (198)
9	School stationary has tobacco-related messages	28% (142)
10	School principal or staff students are awarded for tobacco control activities	23% (112)
11	Availability of advice, consultation from state-appointed tobacco advisor	18% (89)

The dataset, shared by the original researchers [3, 1, 2], included information from 507 schools across 20 districts in Maharashtra. Each school was evaluated based on 11 tobacco-free school (TFS) criteria, with just 11% of schools meeting all criteria, as indicated in Table 1.

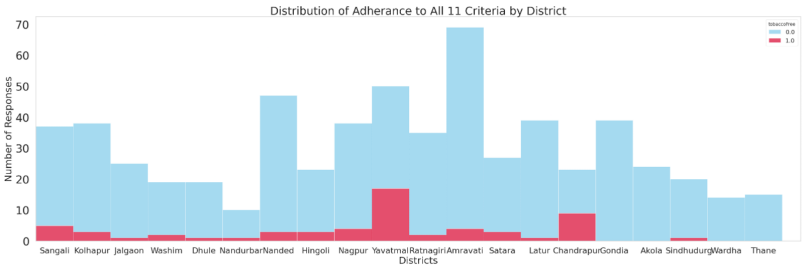

Figure 2: Distribution of Schools Adhering to All 11 Criteria by District (red bars)

One of the early hurdles was simply understanding and reporting the data clearly. Despite multiple reminders, the student failed to produce a table summarizing how many schools met each of the 11 tobacco-free school (TFS) criteria individually, an essential descriptive step that would have grounded the rest of the analysis. Figure 2 illustrates that very few schools met all the requirements, and many met none at all.

When moving into modeling, the student used only 9 features out of over one hundred available in the dataset. These included school-level variables such as participation in sports, internet access, district exam pass rate, infrastructure quality, and teacher training activities. Despite guidance to explore a broader feature set, the limited scope of input data contributed to shallow findings that failed to extend beyond the original study. In addition, although class imbalance was a recurring topic in mentorship meetings, the student did not implement oversampling or other techniques to increase representation of the minority class further constraining model performance.

Three machine learning models were implemented over the course of the project. The first was a logistic regression model, developed using the statsmodels library after both scikit-learn and Firth regression versions failed to converge due to quasi-separation errors. The second was a support vector machine (SVM) with a radial basis function kernel, which produced the highest accuracy, particularly when paired with Factor Analysis of Mixed Data (FAMD) for dimensionality reduction. Finally, the student attempted to build neural network models using both TensorFlow and PyTorch, but these performed poorly and lacked interpretability given the limited dataset and severe class

imbalance. The modeling notes are summarized in Table 2.

Table 2: Feature Usage and Modeling Notes

Model Type	**Features Used**	**Comments**
Logistic Regression	9 selected features	Convergence issues in scikit-learn; statsmodels used due to quasi-separation errors
Support Vector Machine (SVM)	Selected features + FAMD	High reported accuracy driven by severe class imbalance; limited interpretability
Neural Network	Not successfully implemented	Severe class imbalance and lack of sufficient data prevented meaningful model training

The student was also asked to evaluate model results using standard metrics, including confusion matrices, ROC curves, precision, recall, and F1-score. While the SVM model achieved high reported accuracy (up to 97%), this was largely due to class imbalance: only 55 of 507 schools were classified as "tobacco-free." This sparked an important discussion about how to interpret model performance in skewed datasets and why accuracy alone is not a reliable metric.

In one variation, the definition of "tobacco-free" was relaxed from meeting all 11 criteria to meeting 7 or more. This adjustment improved label balance and highlighted how the framing of a prediction target can directly affect model behavior. Although the student didn't fully explore this approach, it served as a useful mentoring moment about the consequences of data definition and preparation. Key differences between the intended mentoring outcomes and observed results are summarized in Table 3.

Table 3: Planned vs. Observed Outcomes in the Mentored ML Project

Planned Expectation	Observed Outcome
Regular independent progress between meetings	Work primarily completed during meetings
Completion of data preprocessing and feature exploration	Limited feature usage and incomplete documentation
Reproducible modeling pipeline	Partial replication with unresolved issues
Submission of polished research paper	Last-minute, incomplete draft

5 Reflections and Conclusion

Documenting unsuccessful undergraduate research experiences constitutes a meaningful scholarly contribution, particularly in teaching-focused institutions. While prior work often highlights successful mentoring models, failure cases expose structural, motivational, and pedagogical gaps that remain invisible in outcome-driven studies. In environments without formal machine learning curricula, these cases provide critical insight into the limits of mentorship, the importance of scaffolding research skills, and the mismatch between student expectations and research demands. By presenting an honest account of a mentoring effort that did not achieve its intended technical goals, this paper addresses an underrepresented gap in computer science education research.

Mentorship is not a guarantee of success, particularly when funding is interpreted as compensation for attendance rather than investment in learning. These cases illustrate how structured mentorship and institutional support fall short when students lack preparation, curiosity, or resilience. While this project didn't lead to a formal outcome, it provided insight into how undergraduate mentoring can fail, and how mentors might recalibrate expectations. Faculty at HSIs, especially at undergraduate-only institutions, must be equipped to identify misalignment early and respond accordingly.

These findings suggest that broadening access to research is insufficient without sustained effort, technical alignment, and mutual accountability from both students and programs. Documenting failure is a necessary contribution to a more honest and effective culture of mentorship. A future mentoring effort is being structured under the Computing Alliance of Hispanic Serving Institutions (CAHSI) and will adopt the Affinity Research Group (ARG) model for guided reflection and goal-setting. This transition reflects the mentor's ongoing learning process in addressing challenges of undergraduate engagement.

References

[1] Nandita Chatterjee et al. "The Tobacco-Free Village Program: Helping Rural Areas Implement and Achieve Goals of Tobacco Control Policies in India". In: *Global Health: Science and Practice* 5.3 (2017), pp. 476–485.

[2] Nandita Chatterjee et al. "Tobacco-Free School Policy in Maharashtra, India: A Qualitative Exploration of Implementation Facilitators and Barriers". In: *Health Behavior and Policy Review* 5.3 (2018), pp. 24–35.

[3] Nandita Chatterjee et al. "Adherence to the Tobacco-Free School Policy in Rural India". In: *Asian Pacific Journal of Cancer Prevention* 18.9 (2017), pp. 2367–2373.

[4] Jill Denner, Linda Werner, and Eloy Ortiz. "Computer Science at a Hispanic-Serving Institution: Gender and Ethnic Diversity". In: *ACM Transactions on Computing Education* 12.4 (2012). Article 16, 16:1–16:17.

[5] I. Garcia Montoya et al. "Incorporating Building Data Pipelines in Data-Engineering Curriculum for Supporting Classification Models in Production". In: *CCSC Mid-South Conference Proceedings*. 2025.

[6] Shobhit Kumar and Lovekesh Vig. "Teaching Machine Learning with Real-World Datasets at the Undergraduate Level". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 2018, pp. 652–657.

[7] E. Lindoo and M. Lotfy. "Generative AI and its Impact on the CS Classroom and Programmers". In: *CCSC Rocky Mountain Conference Proceedings*. 2024.

Generative AI and Software Development Job Security: Challenges and Insights for Computing Education*

Fei Zuo[1], Gang Qian[1], Fang Li[2], Yuqi Song[3], and Xin Zhang[3]
[1]*Department of Computer Science*
University of Central Oklahoma, Edmond, OK 73034
{fzuo, gqian}@uco.edu
[2]*Department of Computer Science*
Oklahoma Christian University, Edmond, OK 73013
fang.li@oc.edu
[3]*Department of Computer Science*
University of Southern Maine, Portland, ME 04104
{yuqi.song, xin.zhang}@maine.edu

Abstract

The transformations brought about by generative AI to the economy and society are beyond people's expectations. In particular, as generative AI is being increasingly applied in the software industry, many computer science graduates are reporting growing difficulty in securing jobs. New technologies can both significantly boost productivity and potentially lead to more unemployment and layoffs. Therefore, the impact of generative AI on the software industry is ambiguous. Limited prior work showed interest in this issue and conducted preliminary explorations. However, existing studies suffer from certain limitations; for example, they often rely on questionnaires or interviews, which may be subjective and even prone to bias. In this paper, we fill this gap by collecting empirical evidence from official statistics and industry reports

and conducting a quantitative analysis. Our empirical evidence reveals the capacity of generative AI to reshape the labor market. The findings also provide valuable implications for computing education.

1 Introduction

In recent years, the booming development of generative AI has brought a transformative impact on various sectors [16, 19]. A landmark event was OpenAI's announcement of their commercial Large Language Model (LLM), ChatGPT, in November 2022. Following this, LLMs quickly garnered widespread interest due to their excellent ability to generate human-like responses in conversational interactions.

Beyond the usage in interactive conversations, LLMs[1] have demonstrated noticeable adeptness in tackling a variety of software engineering problems, such as code generation [5], code refinement [6], and even program repair [11]. Thus, it is widely believed that a certain portion of software development tasks is susceptible to automation. This leads to two-sided consequences. On one hand, generative AI can significantly enhance productivity and efficiency in software development. On the other hand, if the workloads of developers are replaced by generative AI to some extent, it could bring about mass layoffs and unemployment. Not surprisingly, the impact of generative AI on the software industry appears to be contentious and ambiguous.

The economic impact of generative AI on the job market is a topic worthy of in-depth exploration, and we have noticed significant interest from academia in this area. However, our survey indicates that research focusing specifically on job security within software development is still limited. Section 3 provides a detailed analysis of the insufficiency. We seek to fill this gap in the literature.

Our empirical study began by gathering and consolidating data from diverse sources, which formed the foundation of our later discussion. Furthermore, the public release of ChatGPT was treated as a watershed event (equivalent to "exogenous shock", a term in economics) in our research framework, enabling us to observe subsequent trends in both the development of the generative AI industry and employment trends within the software sector. By performing a quantitative analysis, we reveal the challenges that generative AI poses to job security in software development. More importantly, from the perspective of computing education, we share our insights for future curriculum development with the goal of helping students better prepare for ongoing transformations in the industry.

[1] We primarily focus on the role of generative AI in software development, where the generated outputs are textual information including code, documentation, and test cases. So the two terms, generative AI and LLM, are used interchangeably, unless otherwise specified.

2 Article Organization and Research Methodology

This study adopted a dual-method research methodology that integrated a literature review and an empirical data analysis. First, we conducted a systematic survey of existing studies that examined, from an economic perspective, the impact of generative AI on the overall labor market. This included research exploring how automation and AI-driven productivity shifts reshaped workforce demand and the risk of unemployment across industries. Moreover, we reviewed relatively scarce literature that preliminarily explored the influence of generative AI on the software sector. By this means, we obtained a comprehensive understanding of how academia conceptualizes generative AI's potential long-term effects on human labor.

In addition, we took the U.S. software sector as our research context and collected historical data from industry reports and government statistics. Based on this, we conducted a quantitative analysis on both the development of the generative AI industry as well as recent employment trends. This allowed us to test the validity of prior hypotheses regarding the impact of technological transformation on the software labor market. Finally, grounded in historical observations and solid empirical evidence, we derived implications and insights for computing education.

The rest of this paper is organized as follows. We first survey the related literature in Section 3 to identify the current research gap. Section 4 presents the empirical evidence and noticeable findings. We share our insights for computing education in Section 5. Lastly, Section 6 concludes this work.

3 Literature Survey

3.1 Impact on overall labor market

Empirical analyses have already been used to interpret the economic impact of generative AI on the job market. Previous studies indicate that generative AI has a dual impact on reshaping the labor market.

- For jobs that are more easily replaceable or more susceptible to automation, the substitution effect of generative AI is very pronounced. Hui et al. [7] analyzed data from a large online labor market and observed that generative AI leads to a statistically significant decline in both the number of jobs and monthly earnings. Liu et al.k [9] and Chen et al. [4] also independently demonstrated that, for those vulnerable jobs, generative AI has a substantial and increasingly negative effect on employment.

- The emergence of generative AI has also set higher standards for the skill sets that the workforce should possess. Ahmadi et al. [1] stated that

because of the increasing integration of generative AI across various industries, familiarity with this new technology becomes essential in today's labor market. The findings of Chen et al. [4] show that work that can be augmented by generative AI increasingly demands higher skill levels.

This body of work is mainly limited in two aspects. First, the introduction of generative AI is regarded as a static factor, without accounting for changes in its level of development over time. Second, it simply treats the labor market as a homogeneous whole and does not conduct a targeted analysis of the software industry, a sector with its own distinctive characteristics.

3.2 Impact on software industry development

Although scholars have begun to narrow down the scope of research and focus on the impact of generative AI on the software industry, this body of work remains limited in quantity. Sakib et al. [13] argued that the job-displacement effects of generative AI would be highly evident in entry-level occupations involving repetitive tasks for computer science and IT graduates. However, their claim was based solely on anecdotal observations without any supporting evidence.

Other more solid studies were mainly conducted through questionnaires or interviews with practitioners. Through interviewing domain experts, Schedlberger et al. [14] confirmed that generative tasks such as writing documentation, coding, and testing are most susceptible to AI's influence. These tasks happen to make up the main work of most junior or mid-level software developers. Lima et al. [8] also demonstrated that generative AIs have a significant impact on software development. They contended that most developer tasks are susceptible to automation, and consequently "*many tasks would be replaced by or integrated into AIs*". Bonin et al. [2] claimed that the adoption of generative AI among IT professionals could reach up to 97%. They also found that as organizations invest more resources in AI initiatives, concerns about job displacement become increasingly prominent.

These preliminary efforts have mainly relied on qualitative methods, such as interviews with industry practitioners. This carries the risk of the results being subjective or biased. Consequently, the validity of such studies remains limited due to the absence of robust empirical evidence and quantitative analysis.

4 Empirical Evidence

In this section, we first use descriptive evidence to illustrate the recent trends in development in the generative AI industry. Then, we introduce job posting data to describe employment changes in the software industry. Finally, correlation

analysis will quantify the relationship between labor demand and the level of development in the AI industry.

4.1 Development of generative AI industry

A November 2025 report from McKinsey [10] shows that, prior to the release of ChatGPT, the proportion of respondents reporting AI use by their organizations remained roughly stable. However, with the emergence of generative AI, there has been a noticeable increase in the share of organizations using AI. As shown in Figure 1, the adoption of general AI and generative AI among surveyed organizations has surged markedly in tandem since 2023, underscoring the public release of ChatGPT as a watershed event.

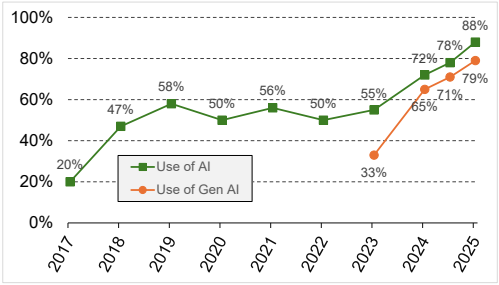

Figure 1: Reported use of AI among organizations continues to increase [10].

The scale of an industry, often reflected by investment growth, is commonly used to quantify its level of development. In a recent report from Ernst & Young [3], researchers studied the index for US real nonresidential fixed investment, and found that traditional business investment remained roughly flat over the past few years, as illustrated in Figure 2. In contrast, AI-related business investments have grown steadily, rising at a 23% annualized pace in the first half of 2025.

In particular, investment in generative AI has continued its rapid ascent. In 2024, the field secured $33.9 billion in funding, marking an 18.7% increase over 2023 and exceeding the 2022 levels by more than 8.5 times [16]. Because code generation offers significant gains in productivity and efficiency, it ranks as the top use case in terms of ROI (Return on Investment), attracting the primary investment focus of organizations [17].

4.2 Observations from the labor market

According to a report by The New York Times [15], a survey of 133 computing programs found that 66% somewhat or strongly agreed that computing majors

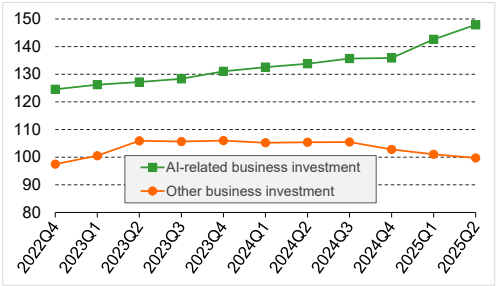

Figure 2: US real nonresidential fixed investment (baseline: 2020 Q1=100).

graduating in 2025 were struggling to secure jobs. This aligns with job-posting data from recent years. As shown in Figure 3, the index of job postings in the software development sector on the Indeed platform has been declining steadily since the fourth quarter of 2022.

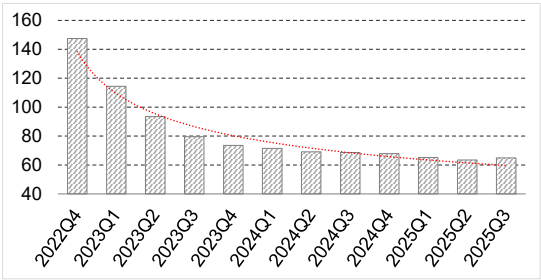

Figure 3: Indeed job postings index (baseline: Feb. 1, 2020=100).

In contrast, the share of job postings on Indeed referencing AI-related terms has risen steadily since mid-2023. As shown in Figure 4, this proportion nearly doubled by the second half of 2025. With employers increasingly seeking talent with AI skills, the volume of such postings are expected to continue growing. This trend aligns with the findings reported by The New York Times [15]: while overall computer science enrollment has begun to decline, enrollment in AI-focused degrees is surging.

4.3 Correlation analysis

We conducted a Pearson correlation analysis to measure the strength and direction of the relationship between the AI industry development level and the employment landscape in the software sector. As Table 1 shows, there is a

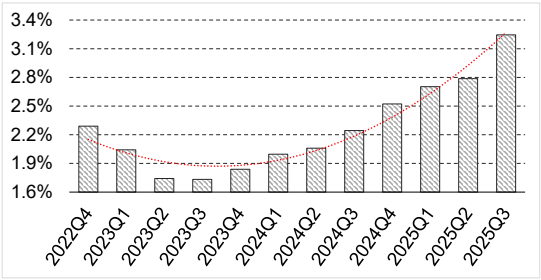

Figure 4: Percentage of job postings containing AI-related terms.

strong negative linear relationship between the AI-related business investment and software development job postings. Moreover, the positive linear relationship between the AI-related business investment and the increasing number of AI-related job positions is also strong. The correlations are statistically significant at the 5% confidence level, based on the corresponding p-values.

Our quantitative analysis indicates that as the AI industry advances, the number of software development jobs gradually declines, whereas AI-related positions continue to grow.

Table 1: The correlation coefficients in our quantitative analysis

	Index of software development job postings	Share of job postings with AI-related terms
Real nonresidential fixed investment related to AI	-0.71851 (p-value: 0.01274)	0.784449 (p-value: 0.00425)

5 Insights for Computing Education

The potential applicability of generative AI has been widely discussed in the context of computing education[5, 12, 20]. The ensuing problems have also sparked controversy. For instance, LLM outputs often lack factual verification, and the much-criticized hallucination problem may mislead students [21]. Furthermore, AI-induced academic integrity issues have posed new challenges for educators . However, previous discussions regarding the impact of the AI industry's growth on the job market has been insufficient and lacks robust empirical substantiation. We argue that this issue deserves greater attention, and share the following insights based on our findings presented earlier.

5.1 Integrating generative AI into existing curricula

There is a growing trend in which colleges across the US are rapidly rolling out new majors centered on AI, and these programs appear to be more appealing to students than traditional computing disciplines [15]. However, we argue that besides establishing new programs, greater effort should be made to integrate generative AI-related topics into existing curricula. To help students better prepare for their future careers, cultivating and strengthening students' human-AI collaboration skills should be highlighted in pedagogical practices.

5.2 Developing comparative advantages over AI

Employees whose job security is disproportionately impacted by generative AI are typically early-career professionals or those with limited technical proficiency. Work tasks that are highly structured or repetitive are particularly susceptible to AI automation, whereas human expertise continues to outperform AI in areas requiring creativity, critical thinking, and complex problem-solving. Therefore, to mitigate the potential disruptions AI may pose to employment, computing education should focus on cultivating students' comparative advantages in these resilient skill sets.

6 Conclusion

Students in computing currently face a challenging job market. While generative AI is likely not the sole contributing factor [18], our quantitative analysis suggests that the industry's increasing adoption of generative AI has exerted a negative pressure on traditional software development positions. Conversely, our results indicate that this adoption has catalyzed growth in AI-related jobs. In response, we provide strategic insights for computing education, advocating the integration of AI into existing curricula and a focus on cultivating skills in areas where AI capabilities remain limited.

Nevertheless, the comprehensive impact of generative AI on the software job market is still unfolding, particularly as the technology continues to evolve rapidly and new LLMs with enhanced capabilities are regularly released by major industry players. Consequently, determining how computing education should adapt to ensure student success in an AI–centric world remains an open question and a critical avenue for future research.

References

[1] Mahdi Ahmadi, Neda Khosh Kheslat, and Adebola Akintomide. "Generative AI Impact on Labor Market: Analyzing ChatGPT's Demand in Job Advertisements". In: *arXiv preprint arXiv:2412.07042* (2024).

[2] Anton Ludwig Bonin, Pawel Robert Smolinski, and Jacek Winiarski. "Exploring the Impact of Generative Artificial Intelligence on Software Development in the IT Sector: Preliminary Findings on Productivity, Efficiency and Job Security". In: *arXiv preprint arXiv:2508.16811* (2025).

[3] Lydia Boussour. *AI-powered growth: GenAI as a new engine of US economic performance*. Accessed: 2025-12. Nov. 2025. URL: `https://www.ey.com/en_us/insights/ai/ai-powered-growth`.

[4] Wilbur Xinyuan Chen, Suraj Srinivasan, and Saleh Zakerinia. *Displacement Or Complementarity? The Labor Market Impact of Generative AI*. Working Paper 25-039. Harvard Business School, 2025.

[5] Paul Denny, Viraj Kumar, and Nasser Giacaman. "Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language". In: *SIGCSE*. 2023, pp. 1136–1142.

[6] Qi Guo et al. "Exploring the potential of ChatGPT in automated code refinement: An empirical study". In: *ICSE*. 2024, pp. 1–13.

[7] Xiang Hui, Oren Reshef, and Luofeng Zhou. "The short-term effects of generative artificial intelligence on employment: Evidence from an online labor market". In: *Organization Science* 35.6 (2024), pp. 1977–1989.

[8] Caroline Da Conceição Lima et al. "Generative AI impact on the future of work: Insights from software development". In: *IEEE SMC*. 2024, pp. 3887–3892.

[9] Yan Liu, He Wang, and Shu Yu. *Labor Demand in the Age of Generative AI: Early Evidence from the U.S. Job Posting Data*. Working Paper 11263. The World Bank, Nov. 2025.

[10] McKinsey & Company. *The state of AI in 2025: Agents, innovation, and transformation*. Accessed: 2025-12. Nov. 2025. URL: `https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai`.

[11] Xianshan Qu et al. "Context Matters: Investigating Its Impact on ChatGPT's Bug Fixing Performance". In: *IEEE/ACIS 22nd International Conference on Software Engineering Research, Management and Applications (SERA)*. 2024, pp. 17–23.

[12] Junghwan Rhee et al. “An evaluation on the impact of large language models on computer science curricula”. In: *Journal of Computing Sciences in Colleges* 40.1 (2024), pp. 70–80.

[13] Nazmus Sakib, Fahim Islam Anik, and Lei Li. “ChatGPT in IT Education Ecosystem: Unraveling Long-Term Impacts on Job Market, Student Learning, and Ethical Practices”. In: *SIGITE*. 2023, pp. 73–78.

[14] Lisa-Maria Schedlberger and Karsten Böhm. “The Essential Human Contribution and its Impact on Future Workforce Development in the Field of Software Engineering”. In: *ICMET*. 2024, pp. 55–63.

[15] Natasha Singer. “College Students Flock to a New Major: A.I.” In: *The New York Times* (Dec. 2025). Accessed: 2025-12.

[16] Stanford HAI. *The AI Index 2025 Annual Report*. Accessed: 2025-12. Apr. 2025. URL: `https://hai.stanford.edu/ai-index/2025-ai-index-report`.

[17] Tim Tully, Joff Redfern, and Derek Xiao. *2024: The State of Generative AI in the Enterprise*. Accessed: 2025-12. Nov. 2024. URL: `https://menlovc.com/2024-the-state-of-generative-ai-in-the-enterprise/`.

[18] Jacob Zinkula. “The year the Big Tech job market cracked”. In: *Business Insider* (Dec. 2025). Accessed: 2025-12.

[19] Fei Zuo, Junghwan Rhee, and Yung Ryn Choe. “Knowledge Transfer from LLMs to Provenance Analysis: A Semantic-Augmented Method for APT Detection”. In: *arXiv preprint arXiv:2503.18316* (2025).

[20] Fei Zuo et al. “ChatGPT as an assembly language interpreter for computing education”. In: *Journal of Computing Sciences in Colleges* 40.2 (2024), pp. 73–82.

[21] Fei Zuo et al. “Revisiting the Capability of GPT in Solving Coding Problems: A Lesson from Programming with Recursion”. In: *Proceedings of the 26th ACM Annual Conference on Cybersecurity & Information Technology Education (SIGCITE)*. 2025, pp. 134–140.

Broadening Participation in Cybersecurity through General Education Pathways*

Timothy Urness
Department of Mathematics and Computer Science
Drake University
Des Moines, Iowa 50311
timothy.urness@drake.edu

Abstract

In this paper, we explore practical strategies for developing and teaching an introductory cybersecurity course designed for a general education audience. This is done on a limited budget, focusing on resources and frameworks that can support small institutions such as liberal arts colleges. We discuss how to leverage existing curriculum frameworks, free and low-cost tools, and online platforms to create impactful learning experiences. This paper provides actionable insights for educators seeking to establish or expand cybersecurity programs while addressing financial and resource constraints.

1 Introduction and Motivation

Introducing a new cybersecurity course or curriculum can feel like a daunting task. Cybersecurity is a dynamic and rapidly evolving field, often perceived as complex and not always well-defined. Even foundational topics can vary widely across introductory resources, with little agreement on the ideal order of instruction or the depth required for each topic. Furthermore, securing funding for networking labs or other technical resources can seem overwhelming,

burdensome, or even unnecessary. These challenges can be particularly intimidating for smaller institutions, such as liberal arts colleges, where technology programs may already be stretched thin. As a result, the idea of launching cybersecurity coursework may feel like an insurmountable endeavor.

Despite these challenges, cybersecurity education is important for a number of reasons.

- There is a growing demand for cybersecurity skills as the need for cybersecurity spans all industries. Developing a cybersecurity program can help make a technology program more "future proof" from unpredictable technology trends.
- Cybersecurity requires a diversity of skills as the subject encompasses a wide range of topics: risk management, networking, programming, policy making, cryptology, operating systems, hardware, etc. A combination of technical and "soft" skills are required to help ensure the security of a company or industry.
- Many cyberattacks exploit human vulnerabilities (e.g., weak passwords, phishing). Education helps individuals recognize and mitigate such risks. Widespread cybersecurity literacy among employees can create a stronger defense against attacks.
- Artificial Intelligence, Internet of Things, and blockchain introduce new cybersecurity challenges that require advanced knowledge to address.
- Cybersecurity is essential to protect national infrastructure, including power grids, financial systems, and military operations.

2 Integrating Cybersecurity into the Liberal Arts Curriculum

At Drake University, we have recently introduced a cybersecurity minor that features a no-prerequisite, non-technical introductory course in cybersecurity. This course was developed as a general-education recruitment tool to entice students to consider studying cybersecurity and is the entry-point to a cybersecurity minor.

As with many liberal art colleges, students are required to take a course from a variety of different categories in order to satisfy general education requirements before graduating. To help incentivize students to consider taking the course, we have developed the cybersecurity course to satisfy the *Information Literacy* category, meaning that students are required to "learn to acquire, analyze, interpret, and integrate information, employing appropriate technology to assist with these processes, and to understand the social and ethical implications of information use and misuse." This is assessed primarily through two individual projects in which annotated bibliographies, papers, and video

presentations are assigned. We feel these research projects help enhance the content of the course and make the material customizable to each individual student's interest.

The introductory course and the curriculum is designed to complement a technology major or any other area of study, with the goal of providing cybersecurity context and establishing a "security mindset." The introductory course has been extremely popular since its inception in 2023 (see figure 1), and has ultimately led to an increasing number of students adopting and graduating with a cybersecurity minor (figure 2).

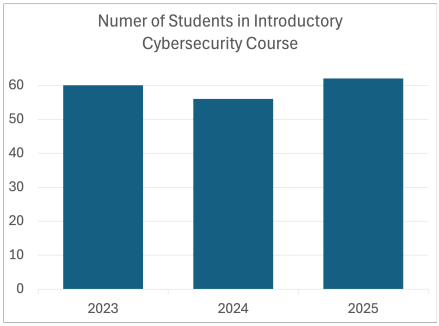

Figure 1: The introductory cybersecurity course has been extremely popular – often filling to capacity.

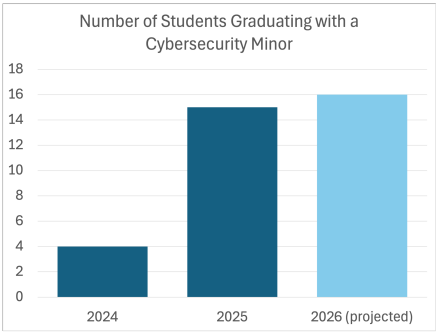

Figure 2: Despite just being introduced in 2023, students have already graduated with the cybersecurity minor.

This paper describes several of the initiatives that have been successful in overcoming hurdles that initially seemed insurmountable or overwhelming.

3 Curriculum Resources and Frameworks

We recognize that each university, program, and student body is distinct, and an approach that succeeds in one context may not translate directly to another. Nonetheless, widely available resources and established frameworks can help guide faculty in building a strong cybersecurity curriculum tailored for their institution. These materials support the introduction of core concepts and the development of skills aligned with industry expectations.

As a practical first step, faculty may benefit from reviewing existing curricular frameworks or exemplars from peer institutions or from the collegiate Advanced Placement board to serve as a foundation for program design.

3.1 AP Cybersecurity

In the 2025-26 school year, various high schools schools will participate in the second pilot for AP Cybersecurity, with the course launching nationally in 2026-27 [1]. The AP Cybersecurity Course Framework is organized into five units that cover different domains of cybersecurity: (1) Introduction to Security, (2) Securing Spaces (physical security), (3) Securing Networks, (4) Securing Devices, and (5) Securing Applications and Data. Furthermore the framework provides learning objectives, example classroom activities in the form of applied, realistic scenarios that can be adopted into in-class activities. These resources are an ideal foundation and starting point.

3.2 CLARK

Cybersecurity Labs and Resource Knowledge-base (CLARK) [4] is a comprehensive online library offering numerous examples of cybersecurity curricula. It serves as a repository where instructors can share resources such as lesson plans, assignments, syllabi, and even entire program designs. According to its website, CLARK is "home to high-value, high-impact cyber curriculum created by top educators and reviewed for relevance and quality." Faculty developing new programs can find inspiration and practical tools on this platform to structure courses effectively.

3.3 NICE Framework

The National Initiative for Cybersecurity Education (NICE) Framework [10], developed by the National Institute of Standards and Technology (NIST), is a key resource for aligning cybersecurity education with professional roles. The framework outlines the knowledge, skills, abilities, and tasks required for various cybersecurity roles. It also includes a "Getting Started" guide to help educators map important cybersecurity competencies to their curricula [11].

4 A General Education Introduction to Cybersecurity Course

While these frameworks provided in the previous section provide an anchor as to what kinds of topics to cover in an introductory course or curriculum, the challenge of how to present this material in an engaging and interactive way to students that may not have yet acquired a passion for the material can be a challenge. Table 1 provides hands-on engaging cybersecurity classroom activities that we have found to be inspiring for students, regardless of any previous exposure to technology or cybersecurity.

Table 1: Cybersecurity Topics and Hands-On Activities

Topic	**Hands-On Activity / Resource**
Social Engineering & Prompt Engineering	Show real-world examples (e.g., Rachel Tobac or Kevin Mitnick youtube videos) and have students try *Gandalf* at `https://gandalf.lakera.ai/gandalf` to explore prompt manipulation.
Internet of Things (IoT)	Investigate IoT device exposure using the web application *Shodan* `https://www.shodan.io/` and *Google Dorking* techniques to understand device discoverability and security risks.
Digital Forensics	Use the web-based *Legends of Learning Digital Forensics* module `https://www.legendsoflearning.com/learning-objectives/cybersecurity-digital-forensics/` to simulate evidence gathering, file recovery, and forensic investigation scenarios without having to install multiple programs to simulate digital forensics.
Networking and Wireshark Lab	Have students download Wireshark and conduct a 45-minute guided lab analyzing captured packets to identify network protocols, detect suspicious traffic, and understand packet structure.
Cryptography	Introduce the students to cyberchef `https://gchq.github.io/CyberChef/`. Have them encrypt and decrypt various codes.
Cryptography / RSA	Have students exchange and decrypt messages using RSA key pairs to explore public-key encryption concepts and implementation. Use online tools or calculators to abstract away as much of the mathematics as appropriate.
Capture the Flag (CTFs)	Participate in beginner-friendly CTFs such as *OverTheWire*, *PicoCTF*, or *OWASP Juice Shop* to apply hacking, scripting, and problem-solving skills.
Risk Assessment	Students develop a cybersecurity insurance company or policy; use a dice-rolling game (e.g. Backdoors and Breaches `https://www.blackhillsinfosec.com/tools/backdoorsandbreaches/`) for a custom table-top exercise
Password Cracking	Demonstrate password vulnerabilities with *Hashcat* or *John the Ripper* using controlled password hashes; discuss ethical and legal implications.
Introduction to Linux / Kali	Use the free *TryHackMe[15] "Introduction to Cybersecurity"* or "Intro to Linux" modules to familiarize students with basic commands, file systems, and security tools.
SQL Injection	Use sample vulnerable web apps (e.g., *DVWA*, *Juice Shop*) to safely practice identifying and exploiting SQL injection flaws.
VPNs	Compare and contrast VPN products, including free browser plug-ins; discuss privacy and performance trade-offs.
Cyber Ethics & Legal Issues	Examine cybersecurity laws and ethical dilemmas using the *Cybersecurity Ethics Case Studies* from the NICE framework or university ethics modules.
Network Defense / Firewalls	Configure basic firewall rules with *ufw* or firewall game `https://cybergamesuk.com/firewall`
Incident Response & Log Analysis	Examine simulated breach logs and develop a simple incident response plan; identify indicators of compromise.

5 The Role of Certifications in Cybersecurity Education

Professional certifications are commonly listed in cybersecurity job postings, underscoring their value in establishing credentials for entry-level employment. While certifications are not universally required, they serve as a means for students to validate their technical competencies and stand out in a competitive job market. Certifications also offer educators a structured framework to identify key concepts and skills that align with industry demands. Additionally, certification study guides can function as course textbooks, and assessments within the curriculum can be designed to reflect the format and rigor of certification exams.

5.1 ISC^2 Certified in Cybersecurity (CC)

The International Information System Security Certification Consortium (ISC^2) "One Million Certified in Cybersecurity" initiative provides free online, self-paced training and certification exams for the Certified in Cybersecurity (CC) credential [7]. This program is particularly accessible for students and institutions seeking cost-effective entry-level certifications. While the training and exams are free, maintaining the credential requires an annual $50 fee. The initiative represents a practical starting point for students or educators to gain foundational cybersecurity skills and credentials.

5.2 CompTIA Certifications

CompTIA provides a widely recognized suite of industry certifications [5]. Although these certifications require a fee to take the exam, the skills and knowledge necessary to prepare for these exams can serve as valuable learning objectives within a collegiate course, regardless of whether students pursue the certification itself. The suite of CompTIA certifications include:

- **Security+**, a foundational certification covering essential cybersecurity concepts and practices.
- **Network+**, which focuses on networking principles and technologies.
- **Cybersecurity Analyst (CySA+)**, emphasizing threat detection and defense.
- **PenTest+**, centered on penetration testing and vulnerability management.

CompTIA certifications can be integrated into academic curricula to provide students with industry-recognized credentials. For example, the CompTIA Security+ certification is well-suited as a foundation for an introductory cybersecurity course. Requiring the Security+ study guide as a textbook [8] and

structuring course assessments to align with the certification exam prepares students for both academic success and credential attainment.

6 Textbooks

A conventional approach to structuring an academic course involves adopting a traditional textbook. Numerous high-quality resources are available to support cybersecurity education, providing foundational knowledge and practical applications.

6.1 Hands-on Security and SEED Labs

One notable example is the textbook series authored by Wenliang Du, which includes volumes on computer security and network security or combined [16]. The textbooks are accompanied by a hands-on labs series (SEED Labs) and videos [17] and provides instructors and students a comprehensive, integrated learning experience.

6.2 ZyBooks

Another textbook option is zyBooks, an interactive, web-based textbook platform that provides dynamic instruction and examples. Relevant zyBooks for cybersecurity courses include Networking [18] and Introduction to Security [19], both of which align closely with CompTIA certification objectives for Network+ and Security+. These zyBooks also feature interactive labs and hands-on exercises, designed to reinforce concepts through practical applications on both Windows and Linux operating systems. Adoption of a zyBook requires a per-student fee, but is typically within the monetary range of a conventional textbook.

6.3 Security in Computing, 6th Edition

Security in Computing, 6th Edition, [3] defines core principles of modern security policies, processes, and protection, offering a foundation in key concepts such as assets, threats, vulnerabilities, controls, and the CIA triad (confidentiality, integrity, availability). This textbook integrates theoretical and practical perspectives, with updated diagrams, exercises, and case studies mapped to key cybersecurity frameworks, including the NICE Framework. The most recent edition also includes expanded coverage of contemporary topics such as artificial intelligence and machine learning in cybersecurity, application and browser security, and strategies for securing cloud, Internet of Things, and embedded systems. While a textbook provides a conventional approach to

classroom resources, this text bridges foundational knowledge with cutting-edge developments, equipping students and professionals to address current and future cybersecurity challenges effectively.

7 Online Resources

The abundance of online resources available on cybersecurity can be both a strength and a challenge for educators. While platforms like YouTube host numerous educational channels, the difficulty lies in identifying high-quality content that aligns with the course's scope and level. For example, searching for prominent figures like Kevin Mitnick or Rachel Tobac can yield valuable content on social engineering. However, it is generally advisable to prioritize established educational resources over ad hoc internet searches to ensure consistency and rigor.

7.1 Tools and Training Platforms

7.1.1 Burp Suite and OWASP ZAP

Burp Suite, developed by PortSwigger, is a widely used web vulnerability scanner and penetration testing toolkit [13]. Its Community Edition is free and provides essential features suitable for beginners, offering a solid foundation for learning web application security. Similarly, OWASP ZAP [12] is a free, open-source tool that provides comparable functionalities, making it an excellent alternative for students and educators. The PortSwigger Academy [14] is a notable free resource, offering interactive labs with step-by-step guidance and instructional videos to complement learning.

7.1.2 pwn.college

Developed by Arizona State University, pwn.college [2] is a free educational platform designed to teach core cybersecurity concepts through hands-on challenges organized into thematic "dojos." These dojos, akin to a karate school, guide learners through progressively challenging exercises delivered via virtual machines. Supplementary recorded lectures from Arizona State University faculty align closely with the coursework, providing a structured learning path for students.

7.2 Competitions and Collaborative Learning

7.2.1 National Cyber League (NCL)

The National Cyber League (NCL) provides a competition-based approach to learning cybersecurity skills [9]. NCL sessions, held in fall and spring, include:

- A gymnasium for practicing concepts with guided examples.
- A practice game, which spans one week and excludes hints or guides.
- An individual game, lasting a weekend.
- A team game, where up to seven participants collaborate to tackle challenges over another weekend.

NCL registration costs $45 for regular entry or $55 for late registration, with free access for coaches supervising registered students. The platform emphasizes real-world, hands-on challenges, making it an excellent supplement to classroom instruction.

7.2.2 HackTheBox and TryHackMe

HackTheBox [6] and TryHackMe [15] are online platforms designed to simulate real-world cybersecurity scenarios through interactive, hands-on exercises. These platforms gamify the learning process, encouraging users to solve challenges such as "capturing the flag" or "hacking the box." TryHackMe is best suited for beginners as it offers structured guidance, walkthroughs, and foundational challenges. HackTheBox also has excellent beginner-oriented walkthoughs, but targets more experienced learners with self-directed, advanced challenges. Both platforms employ tiered pricing models, with free options available for basic challenges and premium features for advanced users.

8 Personal Networks and Collaborations

8.1 Cyber Clubs

Forming a cybersecurity club can foster community and collaboration among students. Clubs provide opportunities to collectively tackle challenges like those on HackTheBox, TryHackMe, or NCL, easing the learning curve often associated with these platforms. Setting goals, such as competing in NCL or progressing through HackTheBox or TryHackMe challenges, can provide structure and motivation for regular meetings.

8.2 Advisory Boards

Nearly all businesses, regardless of size, engage in some level of cybersecurity to protect their digital assets and sensitive information. Engaging local businesses

in establishing a cybersecurity advisory board can provide invaluable insights into the skills and competencies required for the workforce. Semi-annual meetings with the advisory board can help ensure that the curriculum remains aligned with industry needs and that students are equipped with practical, relevant skills for their future roles.

9 Results

In our experience, the introductory, general-purpose, no-prerequisite cybersecurity course has been very successful and has oftentimes filled to the classroom capacity. It is a non-technical course that satisfies a general education component for the liberal arts curriculum and has motivated several students to take the advanced cybersecurity course and complete the cybersecurity minor as introduced earlier in figures 1 and 2.

The original minor introduced in 2023 included the introductory course, an additional, technical "ethical hacking" course, and also utilized existing courses in computer networks, programming, ethics, and discrete mathematics. Upon consultation with the advisory board, who felt the mathematics course wasn't necessary for current trends in cyber, we replaced the mathematics course with another existing course in information technology.

10 Conclusion

Developing a cybersecurity curriculum on a limited budget may seem challenging, but with strategic use of available resources and careful planning, it is both achievable and highly impactful. There is a wealth of affordable or free options that can support the creation of a meaningful program. By leveraging these readily available tools and strategies, institutions can build robust cybersecurity programs that equip students with the skills needed to navigate the complexities of the field and address the growing demand for cybersecurity professionals in an ever-evolving digital landscape.

References

[1] AP Cybersecurity. *About AP Cybersecurity – AP Central | College Board*. URL: `https://apcentral.collegeboard.org/courses/ap-cybersecurity/about`.

[2] Arizona State University. *pwn.college - Free Educational Platform for Cybersecurity Concepts*. URL: `https://pwn.college/`.

[3] Charles P. Pfleeger, Shari Lawrence Pfleeger, and Jonathan Margulies. *Security in Computing, 6th Edition*. Pearson, 2023. ISBN: 978-0137891214.

[4] CLARK Team. *Cybersecurity Labs and Resource Knowledge-base (CLARK)*. URL: https://www.clark.center/.

[5] Computing Technology Industry Association (CompTIA). *CompTIA Certifications*. URL: https://www.comptia.org/certifications/.

[6] Hack The Box Ltd. *Hack The Box - Cybersecurity Training and Hacking Platform*. URL: https://www.hackthebox.com/.

[7] International Information System Security Certification Consortium (ISC²). *One Million Certified in Cybersecurity – Free ISC² Certification Exams*. URL: https://www.isc2.org/landing/1mcc.

[8] Mike Chapple and David Seidl. *CompTIA Security+ Certification Kit (Exam SY0-701)*. Sybex, 2024. URL: https://www.amazon.com/CompTIA-Security-Certification-Kit-SY0-701/dp/1394211449/.

[9] National Cyber League (NCL). *National Cyber League - Cybersecurity Competition and Training Platform*. URL: https://national-cyber-league.super.site/.

[10] National Initiative for Cybersecurity Education (NICE). *NICE Framework: Workforce Framework for Cybersecurity*. URL: https://niccs.cisa.gov/workforce-development/nice-framework.

[11] National Institute of Standards and Technology (NIST). *Getting Started with the NICE Framework Resource Center*. URL: https://www.nist.gov/itl/applied-cybersecurity/nice/nice-framework-resource-center/getting-started.

[12] OWASP Foundation. *OWASP ZAP - Zed Attack Proxy*. URL: https://www.zaproxy.org/.

[13] PortSwigger Ltd. *Burp Suite - Web Vulnerability Scanner and Toolkit*. URL: https://portswigger.net/burp.

[14] PortSwigger Ltd. *Web Security Academy - Free Learning Resources for Web Application Security*. URL: https://portswigger.net/web-security.

[15] TryHackMe Ltd. *TryHackMe - Cybersecurity Training Platform*. URL: https://tryhackme.com/.

[16] Wenliang Du. *Hands-on Security Resources*. URL: https://www.handsonsecurity.net.

[17] Wenliang Du. *SEED Labs: Hands-on Labs for Security Education*. URL: https://seedsecuritylabs.org/labs.html.

[18] zyBooks. *Introduction to Networking with CompTIA Network+ (zyLabs Version 2)*. URL: https://www.zybooks.com/catalog/introduction-to-networking-with-comptia-network-zylabs-version-2/.

[19] zyBooks. *Introduction to Security with CompTIA Security+ (zyLabs Version)*. URL: https://www.zybooks.com/catalog/introduction-to-security-with-comptia-security-zylabs-version/.

Teaching Python Best Practices to Middle School Robotics Students: Curriculum Design and Instructor Reflections*

Abbas Attarwala[1], *Paul R. Viotti*[2], *Mohammed Albahttiti*[3], Hrishikesh Vasant Hasabnis[1]
[1]*Computer Science Department*
California State University, Chico
Chico, CA 95973, USA
{aattarwala,hvhasabnis}@csuchico.edu
[2]*Political Science Department*
California State University, Chico
Chico, CA 95973, USA
pviotti@csuchico.edu
[3]*Concrete Industry Management*
California State University, Chico
Chico, CA 95973, USA
malbahttiti@csuchico.edu

Abstract

Middle school students participating in robotics programs often develop basic Python competency but struggle to reuse their own code across projects due to unclear variable names, duplicated logic, lack of modularity, and missing comments. This experience paper presents a

one-day workshop curriculum designed to address this challenge by introducing code organization practices to grade 7 and 8 students in a LEGO robotics context. Rather than teaching abstract software engineering principles, the workshop uses twenty concrete "bad versus better" code comparisons specifically crafted for this age group. This paper describes the pedagogical rationale behind the instructional examples and explains how contemporary research on erroneous examples and contrasting cases informed the curriculum design. We also reflect on the design decisions that shaped the workshop.

1 Introduction

Middle school represents a unique opportunity in computing education. While many students at this age have little or no prior programming experience, those involved in robotics programs often develop early competency using visual programming environments such as Scratch or LEGO Education SPIKE. As they progress, these learners begin transitioning to text-based languages like Python and are able to write functional programs that include variables, loops, conditionals, and simple functions. Despite these emerging skills, an important challenge frequently goes unaddressed: writing code that works for a single task is not the same as writing code that can be understood, reused, and extended later. Students are often successful at completing an individual robotics mission, but difficulties arise when they encounter a new task that could benefit from earlier work. Consider a student who writes a program for a robot to navigate a maze using a color sensor. The student defines variables, implements conditional logic, and organizes behavior into functions, and the robot successfully completes the maze. Several weeks later, the student is given a new challenge—following a line using the same color sensor. Although much of the original logic remains relevant, including sensor input, decision-making, and motor control, the student struggles to reuse the prior solution. The original code is difficult to read, tightly coupled to the maze task, and poorly structured for modification, making it unclear which components can be adapted to the new problem.

Novice programs in robotics contexts frequently exhibit recurring issues that hinder readability and reuse. Variable names are often incomprehensible, making it unclear whether a variable such as `x` represents a sensor reading or a motor speed, or what a name like `temp` signifies within a given context. Code duplication is also common, with identical sensor-reading logic repeated throughout a program, making it unclear which copy should be modified when behavior needs to change and increasing the risk of unintended side effects. In many cases, programs lack modular structure and are written as single monolithic scripts, preventing learners from easily extracting or reusing

components such as sensor-processing routines. Finally, missing contextual information—such as comments explaining the purpose of code blocks or the rationale behind specific parameter values—further obscures program intent and makes code difficult to understand or extend.

Prior work suggests that novice programmers often start fresh for new tasks rather than building on prior code, particularly when earlier work is difficult to interpret or adapt. In applied computing contexts such as robotics education, learners commonly write Python code that completes a specific mission and then set it aside, instead of adapting it for later tasks, even when substantial portions of the logic remain relevant [3, 8, 4]. When the next competition season arrives, they often start over, not because the previous code is irrelevant, but because it is effectively inaccessible to them. This represents a missed learning opportunity. If learners could effectively reuse and refine code across projects, they would (a) work more efficiently, (b) develop deeper understanding through iterative improvement, and (c) experience programming as a cumulative skill in which each project builds on previous work.

2 The Pedagogical Challenge

The question is not whether middle school students can master abstract software engineering principles, which may be developmentally inappropriate and cognitively overwhelming. Instead, the question is whether learners with basic programming competence can adopt simple, concrete practices that support reuse of their own work. Research in K–12 programming shows that beginners often struggle to build on previous code not solely because of limited ability, but because abstraction-related practices—such as decomposition, generalization, and structuring—are rarely taught explicitly or scaffolded in age-appropriate ways, limiting students' ability to understand, adapt, and transfer prior solutions [14].

Many curricula treat code organization and quality as concerns that can be postponed until students have become fluent in basic programming. This belief is reflected in the way that introductory materials focus almost exclusively on producing working programs while largely ignoring structure, readability, and reuse [9]. However, ignoring these practices has consequences. Research shows that students rarely fix code quality issues as they continue developing their programs [10], suggesting that patterns established early may persist if not explicitly addressed. This pattern aligns with studies showing that novices often abandon functioning code and reimplement solutions from scratch because previous work is too disorganized or inaccessible to reuse [4, 3, 8].

An alternative approach is to introduce basic organizational practices early in learners' programming experiences. Previous work shows that young learn-

ers can meaningfully engage with simple code quality ideas when these ideas are tied directly to their own work. Meaningful variable names can support conceptual understanding [16]. Lightweight procedural abstractions can reduce cognitive load by helping students encapsulate repeated patterns [15, 13]. Scaffolded supports for simple refactoring have enabled beginners to improve the structure of their Scratch programs without requiring advanced expertise [6]. Strategic commenting has also been linked to improvements in the way students reason about and revisit their code [13]. In all of these studies, the consistent finding is that early exposure to concrete organizational practices can enhance foundational programming instruction rather than compete with it.

This paper contributes to the computing education community in two ways: (1) We present a curriculum of twenty "bad versus better" Python examples specifically designed for middle school robotics students, with detailed rationale for why each practice benefits learners at this stage. (2) We offer practical reflections on curriculum design, discussing which pedagogical approaches proved most effective, and providing guidance for educators who wish to adapt this material for their own contexts.

3 Theoretical Foundations for Curriculum Design

The workshop curriculum draws on three interconnected bodies of research: learning from erroneous examples, contrasting case pedagogy, and situated learning theory.

[5] report that learning from erroneous examples is most effective when learners are explicitly guided to compare flawed and correct solutions, helping them construct what the authors term *negative knowledge*, an understanding of why certain problem-solving paths fail. This finding aligns with our decision to present each concept as a pair of "bad versus better," rather than showing only correct solutions. By having learners observe a typical incorrect pattern (e.g., unnecessary variables, repeated print statements, or over-complicated boolean conditions) and then immediately contrast it with a cleaner alternative, we create opportunities for learners to articulate why the first version is problematic. [2] found that erroneous worked examples can enhance learning when the embedded errors are simple enough for novices to detect and correct. This guided our selection of errors: we chose issues that middle school learners with basic Python experience would recognize as problems once pointed out, such as single-letter variable names or repeated code blocks.

[12] demonstrated that elementary learners who studied side-by-side examples of ineffective and effective code achieved significantly higher programming performance and learning engagement than those exposed only to standard worked examples. These results provide empirical support for the "bad versus

better" format used throughout our curriculum. The side-by-side presentation serves multiple instructional purposes: it makes contrasts visually salient, reduces the cognitive burden associated with mentally holding and comparing two versions of a solution, and creates natural opportunities for discussion about why one version improves upon the other.

From a learning-theoretic perspective, the curriculum aligns with principles of situated learning and self-regulated learning. By embedding instruction in authentic robotics programming tasks, learners engage with Python practices as tools for participation in a meaningful activity system rather than as abstract rules [11]. The emphasis on code organization, commenting, and modularity supports self-regulated learning by externalizing planning, monitoring, and reflection processes that enable students to revisit and adapt their own prior work [17]. The use of side-by-side comparisons encourages students to analyze and evaluate alternative representations of the same solution, moving beyond simple code execution toward higher-order cognitive processes such as evaluation and revision [1, 7].

4 Workshop Context and Structure

The workshop was conducted with middle school learners in grades 7–8 (approximately ages 12–14) enrolled in a LEGO Education SPIKE robotics program at a partner school. Participants were not absolute beginners; the workshop assumed prior exposure to block-based robotics programming and introductory Python. The workshop targets organizational practices that are often not emphasized in early robotics programming instruction but strongly support code reuse. Prior experience in similar settings often emphasizes completing individual projects or competition missions, with each project treated as an isolated effort.

The workshop was structured as a single day of instruction and progressed through twenty examples at a pace that allowed time for discussion of each "bad versus better" pair. For each example, instruction followed a consistent pattern: the session began with a poorly written version to surface potential issues, followed by a revised version that made the improvement explicit. Whenever possible, the practice was then connected to robotics programming tasks, followed by a brief explanation of why the practice supports readability, modification, and reuse.

Table 1 presents all twenty practices covered in the workshop, organized into four thematic categories. For each practice, we identify what foundational knowledge it builds upon, why it is appropriate for early introduction, and how it benefits learners in robotics contexts.

Table 1: Twenty Python Best Practices: Rationale and Benefits for Middle School Learners

Practice	Builds Upon	Why Introduce Early	Learner Benefit	Robotics Application
Category 1: Naming and Readability				
Clear variable names	Basic variable assignment	Variables are already familiar; naming is immediately accessible	Code becomes self-documenting; easier to revisit	Sensor readings, motor speeds become identifiable
Descriptive function names	Basic function definition	Extends variable naming to functions	Functions communicate purpose without reading body	`turn_right()` vs. `f1()`
Constants vs. magic numbers	Variable assignment	Requires no new syntax; only naming convention	Values gain meaning; changes propagate automatically	Sensor thresholds, speed limits documented
Meaningful loop variables	Basic for-loop syntax	Natural extension of variable naming	Nested loops become readable	Iterating over sensors, motors
Category 2: Code Structure and Modularity				
Loops instead of repetition	Scratch repeat blocks	Direct transfer from visual programming	Code becomes shorter and modifiable	Repeated motor commands consolidated
Functions for reuse	Scratch custom blocks	Extends familiar concept to Python	Logic extracted for multiple missions	`follow_line()` reused across seasons
Lists for grouped data	Basic variable use	Replaces numbered variables	Data scales without code changes	Multiple sensor readings stored together
Range with steps	Basic range() use	Extends known function	Counting patterns simplified	Motor speed increments
Default parameters	Basic function parameters	Small addition to known concept	Functions become more flexible	Optional speed/duration arguments
Avoiding deep nesting	Conditional statements	Simplifies existing knowledge	Logic becomes clearer	Sensor threshold checks combined
Category 3: Documentation and Formatting				
Writing comments	None required	No syntax to learn; immediate benefit	Code intent preserved for future self	Explaining why specific values chosen
Blank lines between functions	Function definition	Visual organization only	Code gains structure	Separating mission functions
Breaking long lines	String basics	Python-specific but simple	Code fits on screen	Long print messages readable
F-strings for output	String concatenation	Cleaner than + operator	Output code more readable	Debugging sensor values
String join() method	Lists and loops	Replaces manual concatenation	More Pythonic code	Building status messages
Category 4: Python Idioms and Error Prevention				
Boolean values directly	Boolean variables	Removes redundancy	Cleaner conditional logic	`if is_running:` vs. `if is_running == True:`
Handling division by zero	Division operator	Prevents common crash	Programs more robust	Averaging sensor readings safely
Avoiding unused variables	Variable assignment	Reduces clutter	Code stays minimal	Removing debugging artifacts
Import best practices	Using libraries	Relevant to SPIKE imports	Dependencies clear at top	`from spike import Motor` organized
Loops with lists	Lists and loops	Combines two known concepts	Iteration pattern established	Processing multiple sensor values

5 Selected Examples with Full Rationale

Table 1 summarizes all twenty examples; this section presents seven examples in detail, organized by category. All the twenty examples covered in the workshop are available on our GitHub [1]. For each example in this Section, we provide the "bad" and "better" code versions along with implementation-focused design reflections.

5.1 Category 1: Naming and Readability

5.1.1 Clear Variable Names

Variable naming is a foundational practice affecting code readability and was deliberately placed first in the workshop sequence. Research shows that meaningful variable names support conceptual understanding in novice programmers [16].

Table 2: Example of Variable Naming in Robotics Competition Code

Bad

```
x = 75
y = 2
z = x * y
motor.run(z)
```

Better

```
base_speed = 75
speed_multiplier = 2
motor_speed = base_speed *
    ↪ speed_multiplier
motor.run(motor_speed)
```

Introducing clear variable naming early requires no new syntax, only a shift in habit. Because learners at this level typically already know how to create variables, the practice is immediately accessible; novice programs commonly rely on single-letter variable names, as shown in the first column of Table 2. In that version, variables such as `x`, `y`, and `z` obscure the intent of the computation, forcing the reader to infer meaning from surrounding code. By contrast, the revised example uses descriptive names such as `base_speed`, `speed_multiplier`, and `motor_speed`, allowing the code to become largely self-documenting. The example is intentionally constructed so that the poorly named version forces the reader to infer meaning from surrounding code, whereas the revised version communicates intent directly through naming. This contrast highlights how vague variable names can hinder understanding, debugging, and safe modification in robotics programs.

[1] `https://github.com/attarwal/pythonworkshop/tree/main`

5.1.2 Using Constants Instead of Magic Numbers

Magic numbers—unexplained numeric literals embedded directly in code—make programs difficult to understand and modify. In the sensor-based example shown in Table 3, the value 10 in the first column provides no indication of why that particular distance was chosen or what it represents. Replacing such values with named constants documents their meaning explicitly and ensures that changes propagate consistently throughout the program.

Table 3: Using Named Constants Instead of Magic Numbers in Robotics Code

Bad

```
if distance_sensor.
    ↪ get_distance_cm()
    ↪  < 10:
     motor_pair.stop()
```

Better

```
MIN_SAFE_DISTANCE_CM = 10

if distance_sensor.get_distance_cm
    ↪ () < MIN_SAFE_DISTANCE_CM:
     motor_pair.stop()
```

Importantly, introducing this practice requires no new syntax beyond what learners already know about variables; the use of an all-caps naming convention is easy to adopt and clearly signals that a value is intended to remain fixed. This approach is particularly beneficial in robotics contexts, where sensor thresholds and motor speed limits are frequently adjusted during testing and competition. By naming a constant such as `MIN_SAFE_DISTANCE_CM`, learners can modify system behavior by changing a single definition rather than searching through code for every occurrence of a numeric literal. This not only reduces errors but also supports iterative experimentation. This example further shows the value of the practice in a robotics setting. The numeric literal in the original code provides no indication of its purpose or units, making the intent of the threshold ambiguous. The all-caps naming convention provides a simple visual cue that distinguishes fixed configuration values from variables that change during execution.

5.2 Category 2: Code Structure and Modularity

5.2.1 Using Loops Instead of Repetition

Repeated code is one of the most common issues in novice programs. This example introduces the DRY (Don't Repeat Yourself) principle in an accessible way that connects directly to learners' Scratch experience.

Introducing loops early builds naturally on learners' prior experience with visual programming environments such as Scratch, where repeat blocks are used extensively. The Python `for` loop shown in Table 4 serves as a direct

Table 4: Replacing Repeated Robotics Commands with a Loop

Bad	Better

`motor.run(50)`
`wait_for_seconds(1)`
`motor.run(50)`
`wait_for_seconds(1)`
`motor.run(50)`
`wait_for_seconds(1)`
	`for _ in range(3):`
	`  motor.run(50)`
	`  wait_for_seconds(1)`

textual translation of Scratch's repeat block, making the transition from visual to text-based programming straightforward and reducing cognitive load. Beyond reducing code length, loops support abstraction by making programs easier to modify and scale. In the robotics example, increasing the number of repeated motor commands from three iterations to one hundred requires changing a single numeric value, rather than duplicating or rewriting code. This pattern extends naturally to common robotics tasks such as repeated motor actions, sensor polling, and calibration routines, highlighting the practical value of abstraction in robotics contexts.

5.2.2 Using Functions for Reuse

Functions are central to code reuse, which is the workshop's core goal. This practice enables learners to extract logic that can be used across multiple robotics missions.

Table 5: Using Functions to Encapsulate Repeated Robotics Actions

Bad	Better

`left_motor.run(60)`
`right_motor.run(60)`
`wait_for_seconds(2)`
 	
`left_motor.run(40)`
`right_motor.run(40)`
`wait_for_seconds(2)`
	`def drive(speed, duration):`
	`  left_motor.run(speed)`
	`  right_motor.run(speed)`
	`  wait_for_seconds(`
	`  ↪ duration)`
	
	`drive(60, 2)`
	`drive(40, 2)`

Introducing functions early builds naturally on learners' prior experience with visual programming environments such as Scratch, where custom blocks ("My Blocks") serve an analogous purpose. In Table 5 above, the repeated motor-control sequence in the first column is encapsulated in a single `drive(speed, duration)` function, showing how lightweight procedural abstraction can reduce cognitive load by allowing learners to reason about what the robot should

do rather than how each action is implemented. Because many learners are already familiar with grouping repeated instructions for reuse, this transition builds on existing conceptual knowledge and supports the early development of modular thinking [15]. This abstraction enables learners to construct reusable building blocks that persist across different competition missions. Functions such as `drive`, `turn_right`, or `follow_line` can be reused and recombined in new contexts, shifting learners' focus from writing disposable scripts to developing a personal library of reliable components. Framing functions as reusable units emphasizes how motor-control and sensor-processing routines can be encapsulated and reused across robotics missions.

5.2.3 Using Lists for Grouped Data

Lists replace the common novice pattern of creating numbered variables (`sensor1`, `sensor2`, etc.) with a scalable data structure.

Table 6: Grouping Sensor Data Using Lists in Robotics Code

Bad

```
sensor1 =
    ↪ color_sensor_left.
    ↪ reflection()
sensor2 =
    ↪ color_sensor_right.
    ↪ reflection()
sensor3 = distance_sensor.
    ↪ get_distance_cm()
```

Better

```
sensor_values = [
    color_sensor_left.reflection()
        ↪ ,
    color_sensor_right.reflection
        ↪ (),
    distance_sensor.
        ↪ get_distance_cm()
]
```

Introducing lists early builds on learners' existing understanding of variables by extending the concept from single values to collections of related data. In Table 6 above in the first column, the use of separate variables (`sensor1`, `sensor2`, and `sensor3`) shows a pattern that is common in novice robotics programs, but which quickly becomes difficult to manage as systems grow. The presentation of the list-based alternative makes the limitations of numbered variables immediately apparent and highlights how grouping related values can simplify the program structure. This approach allows data to scale without requiring structural changes to the code. In the list-based version of the example, additional sensor readings can be incorporated by adding a single item to the list, rather than introducing new variables and updating every location where sensor data are referenced. In robotics contexts, this pattern naturally applies to managing multiple sensor readings, motor configurations, or collections of mission parameters. This example highlights the benefit of replacing numbered variables with a scalable structure. Describing lists as containers for related items clarifies how this structure addresses common scalability challenges in

robotics programs. As projects grow in complexity, managing numbered variables becomes increasingly unmanageable, whereas list-based approaches scale naturally.

5.3 Category 3: Documentation

5.3.1 Writing Comments to Explain Code

Strategic commenting supports learners' reasoning about their code and encourages reflection on prior solutions [13]. Comments preserve the intent behind code decisions that may not be obvious from the code itself.

Table 7: Using Descriptive Variable Names and Comments in Robotics Calculations

Bad

```
x = 5.6
y = 3.14
z = x * y
```

Better

```
# Wheel diameter in centimeters
wheel_diameter_cm = 5.6
# Approximate value of pi
PI = 3.14
# Calculate wheel circumference
wheel_circumference_cm = wheel_diameter_cm *
    ↪ PI
```

Introducing comments early requires no additional programming syntax and builds naturally on learners' existing ability to write explanatory text. In the example above, brief comments clarify the purpose of each value and computation, making it immediately apparent how the wheel diameter and the constant `PI` contribute to calculating wheel circumference. This benefit becomes especially clear when learners consider revisiting uncommented code weeks later, when the original reasoning behind specific values or calculations may no longer be obvious. Comments preserve contextual information that code alone cannot convey. In robotics programs, numeric values often represent design decisions—such as physical dimensions, sensor thresholds, or calibration parameters—that are not self-evident from the computation itself. By documenting why a particular value was chosen or what a sequence of motor commands is intended to accomplish, comments support both individual reflection and collaborative work, answering questions that a future version of the learner or a teammate will inevitably ask. Table 7 emphasizes why commenting is an essential companion to meaningful naming. The example intentionally combines meaningful variable names with explanatory comments to highlight their complementary roles, emphasizing that comments should explain why code exists rather than merely restating what it does. The example

underscores how easily comments are omitted in novice code and why making commenting an explicit practice is important.

5.4 Category 4: Python Idioms and Organization

5.4.1 Import Best Practices

Proper import organization affects code maintainability and is particularly relevant for SPIKE robotics code, which begins with imports like `from spike import PrimeHub, Motor`.

Table 8: Organizing Imports in Robotics Programs

Bad

```
print("Starting robot")

import math
distance = math.sqrt(100)

from math import *
turn_angle = acos(0.5)

def compute_arc_length(
    ↪ radius):
    from math import pi
    return 2 * pi * radius
```

Better

```
# All imports at the top
from math import sqrt, acos, pi

print("Starting robot")
distance = sqrt(100)
turn_angle = acos(0.5)

def compute_arc_length(radius):
    return 2 * pi * radius
```

Introducing import best practices early supports a clear mental model of a program's external dependencies before code complexity increases. In Table 8's first column, mathematical functions are imported in multiple locations throughout the program in the left column, including inside a function definition, making it difficult to determine which libraries are required and where specific names originate. Consolidating imports at the top of the file, as shown in the revised version, makes dependencies explicit and immediately visible to the reader. This practice requires no new programming concepts beyond those learners already use when importing libraries, but emphasizes organization and clarity. The benefits of this practice are particularly evident in robotics programs, where learners frequently rely on mathematical functions for distance calculations, turning geometry, or motion planning. By importing only the required functions at the top of the file, learners can more easily understand how sensor readings and movement calculations are supported by external libraries. This organization also reduces confusion caused by wildcard imports and prevents subtle errors that arise when names are introduced implicitly or scoped unpredictably. This example highlights why the convention matters as robotics scripts grow in size. In the poorly structured version of the code, it is difficult to determine where specific functions such as `acos` or constants like `pi`

originate. Placing all imports at the top makes program dependencies explicit and improves readability as robotics scripts grow longer.

6 Pedagogical Reflections

Based on the design and implementation of this curriculum, we offer the following reflections for educators considering similar instruction.

The "bad versus better" contrast was designed to promote discussion around code quality and to make common issues easy to notice. The contrasting pairs make common code-quality issues visible and clarify why the "better" version improves readability and reuse. This approach aligns with prior research indicating that contrasting cases can improve learning outcomes [12]. Connecting new Python practices to learners' existing knowledge in Scratch also played an important role in supporting comprehension. The Python constructs were explicitly linked to familiar Scratch elements—for example, loops corresponding to repeat blocks and functions serving a role similar to custom blocks. This linkage leverages familiar concepts from block-based environments, reducing perceived novelty and allowing the practices to build on established mental models.

Beginning instruction with variable naming proved to be an effective entry point. The concept was immediately accessible and directly connected to learners' own programming experiences. Single-letter naming is common in novice code, and its readability costs become apparent once contrasted with descriptive naming. Finally, situating these ideas in a robotics context provided strong motivation by emphasizing their practical value for adapting code across tasks, rather than rewriting solutions from scratch.

Differences in prior programming experience required careful differentiation during the workshop. Instructors should anticipate variation in prior Python experience: some learners may find certain examples straightforward, while others may benefit from additional explanation and guided practice. Future iterations of the workshop could address this range of experience by incorporating tiered examples or optional extension challenges that allow more advanced students to deepen their engagement without leaving others behind. Some examples assume prerequisite knowledge that may not be uniformly present in middle school robotics settings. In particular, the `join()` string method and certain Python idioms may be unfamiliar at this level. These examples were most effective when presented as opportunities for exposure rather than as concepts requiring immediate mastery, allowing students to recognize patterns without the pressure of full understanding. Finally, balancing simplicity with authenticity proved challenging. The examples were intentionally simplified to fit within the constraints of the workshop format, whereas real robotics

programs often involve greater complexity and interdependence. Supporting students in transferring these practices to longer, more realistic programs may require additional scaffolding, such as revisiting the practices in subsequent sessions or embedding them within extended robotics projects.

7 Conclusion

This paper has presented a curriculum for teaching Python best practices to middle school learners in robotics contexts. The twenty "bad versus better" examples are grounded in research on erroneous examples and contrasting case pedagogy, and are designed specifically for learners who have basic Python competency but lack explicit instruction in code organization.

The central pedagogical argument of this work is that code organization should not be deferred until students become advanced programmers. Learners who are able to write functional code can also learn simple organizational ideas that make their programs understandable and reusable. Introducing these ideas early within authentic robotics tasks supports the development of habits that frame programming as a cumulative activity, in which new projects build on existing code rather than replace it.

Acknowledgment

We gratefully acknowledge the use of OpenAI's ChatGPT for proofreading, grammatical checks, and other text editing tasks.

References

[1] Lorin W. Anderson and David R. Krathwohl. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman, 2001.

[2] Maik Beege et al. "Learning programming from erroneous worked-examples. Which type of error is beneficial for learning?" In: *Learning and Instruction* 75 (2021), p. 101497.

[3] Luca Chiodini. "Teaching Introductory Programming Using Graphics as a Domain". English. PhD thesis, USI Faculty of Informatics. Doctoral thesis. Università della Svizzera italiana, 2025. URL: `https://n2t.net/ark:/12658/srd1332702`.

[4] Luca Chiodini, Joey Bevilacqua, and Matthias Hauswirth. "The Toolbox of Functions: Teaching Code Reuse in Schools". In: *Proceedings of the 6th European Conference on Software Engineering Education*. 2025, pp. 185–189.

[5] Sonja Dieterich, Stefan Rumann, and Marc Rodemer. "Conditions for Effective Learning from Erroneous Examples: A Systematic Review". In: *Educational Psychology Review* 37.4 (2025), p. 94.

[6] Paul A Gross et al. "A code reuse interface for non-programmer middle school students". In: *Proceedings of the 15th international conference on Intelligent user interfaces*. 2010, pp. 219–228.

[7] Diane F. Halpern. *Thought and Knowledge: An Introduction to Critical Thinking*. 4th. Mahwah, NJ: Lawrence Erlbaum Associates, 2003.

[8] Ricardo Hidalgo Aragón, Jesus M. Gonzalez-Barahona, and Gregorio Robles. "How Do Code Smells Affect Skill Growth in Scratch Novice Programmers?" In: *SSRN Electronic Journal* (July 2025). Preprint, pp. 1–7. DOI: 10.2139/ssrn.5363087. URL: https://ssrn.com/abstract=5363087.

[9] Cruz Izu et al. "Introducing code quality at cs1 level: Examples and activities". In: *2024 Working Group Reports on Innovation and Technology in Computer Science Education*. 2025, pp. 339–377.

[10] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. "Code quality issues in student programs". In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. 2017, pp. 110–115.

[11] Jean Lave and Etienne Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge, UK: Cambridge University Press, 1991.

[12] Ning Ma et al. "Promoting programming education of novice programmers in elementary schools: A contrasting cases approach for learning programming". In: *Education and Information Technologies* 28.7 (2023), pp. 9211–9234.

[13] Florian Obermüller et al. "Code perfumes: Reporting good code to encourage learners". In: *Proceedings of the 16th Workshop in Primary and Secondary Computing Education*. 2021, pp. 1–10.

[14] Annika Oser and Bernhard Standl. "Teaching and Assessing Abstraction in K-12 Computational Thinking Education: A Systematic Literature Review". In: *Computer Applications in Engineering Education* 33.5 (2025), e70073.

[15] Simon P. Rose, MP Jacob Habgood, and Tim Jay. "Designing a programming game to improve children's procedural abstraction skills in scratch". In: *Journal of Educational Computing Research* 58.7 (2020), pp. 1372–1411.

[16] Peeratham Techapalokul and Eli Tilevich. "Understanding recurring quality problems and their impact on code sharing in block-based software". In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2017, pp. 43–51.

[17] Barry J. Zimmerman. "Self-regulated learning and academic achievement: An overview". In: *Educational Psychologist* 25.1 (1990), pp. 3–17.

Cross-Cultural Challenges and Learning in a COIL-Based Software Engineering Collaboration*

Olivera Grujic[1], Paulo Picota[2]
[1]*Department of Computer Science*
California State University, Stanislaus
Turlock, CA
ogrujic@csustan.edu
[2]*Department of Computer Programming*
Universidad Tecnológica de Panamá
Panamá City, Panamá
paulo.picota@utp.ac.pa

Abstract

This paper reports on a six-week Collaborative Online International Learning (COIL) module connecting an undergraduate Software Engineering course at a broad-access U.S. university with a frontend development course at a selective technical institution in Panama. Although COIL is often presented as a mechanism for enhancing global collaboration and student engagement, our findings show that the effectiveness of such experiences depends heavily on local course culture, student preparedness, and reliable teamwork.

Quantitative survey data (47 U.S. and 12 Panama students) reveal high willingness among Panama students to engage in future COIL experiences but significant frustration driven by inconsistent responsiveness, missed deadlines, and incomplete backend deliverables from their U.S. partners. U.S. students, many with uneven preparation, struggled

with communication, scheduling, and code integration. The collaboration functioned only because instructors established strict milestones and actively coordinated progress across both institutions. We argue that COIL can amplify existing weaknesses in broad-access computing environments.

1 Background and Context

Collaborative Online International Learning (COIL) initiatives aim to expose students to cross-cultural collaboration and distributed teamwork through shared course modules across institutions. In practice, however, COIL implementation varies dramatically depending on local academic structures, faculty coordination, and student preparation. Prior research has shown that cross-cultural and virtual exchange experiences can improve intercultural competence and communication skills [1, 3].

In this project, the partnership involved a frontend Web Design course at a selective engineering university in Panama and a backend-focused Software Engineering course at a regional, broad-access public university in the United States. A prior experience report from this COIL module analyzed only the U.S. cohort due to missing survey data from Panama [2]. In this paper we revisit the same collaboration after delayed Panama responses became available and present a direct comparison between the two institutions. The students differed substantially in academic preparation, prior programming experience, and expectations of teamwork. Panama students tended to display consistent motivation, clearer planning behaviors, and stronger accountability norms. In contrast, the U.S. cohort showed uneven foundational preparation, particularly in backend programming and collaboration.

2 Implementation

The two courses overlapped for only six weeks due to mismatched academic calendars. The only mutually available synchronous time was a single 75-minute joint meeting each Tuesday. Despite these constraints, instructors worked intensively to make the collaboration viable. The U.S. instructor provided the Canvas and Zoom infrastructure, taught one synchronous lecture (on Java Server Pages), reviewed weekly progress reports, and supported backend integration. The Panama instructor delivered three synchronous lectures on HTML, CSS, and Web Forms. Both instructors coordinated around scheduling and institutional constraints.

The COIL module followed a compressed six-week hybrid sprint model, embedded inside a 15-week U.S. semester. Prior to the beginning of the COIL

overlap, the U.S. students completed their documentation phase and entered a four-week coding phase, adopting a lightweight Agile-like approach with weekly milestones and peer accountability structures. The Panama students, whose course emphasized web design, began contributing frontend assets a week prior to the beginning of the COIL module. Four weeks were designed for lectures and coding, one week was intentionally left for integration, and the last week was saved for final presentations and demos. Final presentations were attended by several Panamanian faculty and recorded.

Eight cross-institutional teams were formed. Each team was allotted 8 minutes (due to time constraints) to demonstrate completed or partially completed systems. The U.S. students were responsible for coding the application, while Panama students were given the opportunity to improve upon the appearance of the corresponding web site. The U.S. course enrolled 47 undergraduate students across two sections, while the Panama course enrolled 40 students. Teams remained stable throughout the six-week collaboration period. Additional details on the COIL design and U.S.-only findings are discussed in a companion paper [2].

Six teams delivered complete software projects, but only four successfully integrated Panama-built front-end interfaces with U.S. backend components. Two teams were unable to integrate due to late or nonfunctional backend deliverables on the U.S. side. Despite significant hurdles including limited overlapping schedules and inconsistent participation from some U.S. students, the collaboration remained functional through continuous instructor coordination.

3 Methods and Findings

Both institutions administered a shared six-item Likert-scale instrument (Q1-Q6) plus an open-ended reflection (Q7). The full item text is included in Table 1.

U.S. students completed the survey in class during the final exam, yielding a complete dataset (N=47). Panama students completed an optional online survey near the end of their term (N=12), reflecting a smaller and more selective subset of perspectives. Because the Panama survey was optional, responses may disproportionately reflect the perspectives of more engaged or motivated students. All U.S. survey procedures were exempt by the Institutional Review Board. Participation was voluntary for students in Panama.

Likert items were mapped to numerical values (1-5) to calculate mean scores and compare response patterns across institutions. All qualitative responses were coded inductively into five shared thematic categories: Positive Emotion, Collaboration, Cultural Awareness, Learning/Skills, and Challenges. This coding scheme mirrors the one used in the earlier U.S.-only analysis of the same

Table 1: COIL Survey Questions (Q1–Q7)

Q#	Survey Question
Q1	Before starting COIL project, I was concerned with my ability to collaborate with students from another country in a classroom setting.
Q2	The COIL project provided me with meaningful opportunity to communicate with students from another country.
Q3	The COIL project provided me with meaningful opportunity to collaborate on a project (assignment) with students from another country.
Q4	I am willing to participate in another international program or project.
Q5	I am willing to take another course that connects me with students from other countries.
Q6	The platforms (Canvas, Zoom, Panopto, etc.) used for the virtual classroom meetings were supportive.
Q7	COIL Reflection Journal: Please reflect (comment) on your experience with the COIL course. What did you think?

course [2], enabling direct comparison.

3.1 Quantitative Comparison (Q1–Q6)

Figure 1 presents average Likert scores for both institutions. Several patterns emerged:

- **Q1: Initial concern.** U.S. students reported a slightly higher mean concern (2.72) than Panama students (2.50). The difference is small, but it suggests that more U.S. students anticipated difficulty at the outset, even though several later underestimated the sustained coordination required.
- **Q2: Communication.** Panama students reported lower satisfaction with communication (mean 3.50) compared to U.S. students (mean 3.79). Reflections from Panama frequently mention unmet expectations around predictable responses and timely updates, whereas several U.S. students perceived communication as "fine" despite inconsistent responsiveness.
- **Q3: Collaboration.** Both groups evaluated collaboration positively, but Panama students reported a notably higher mean (4.42) than U.S. students (3.85). This indicates that Panama students valued the collaborative concept strongly, even when execution challenges occurred.

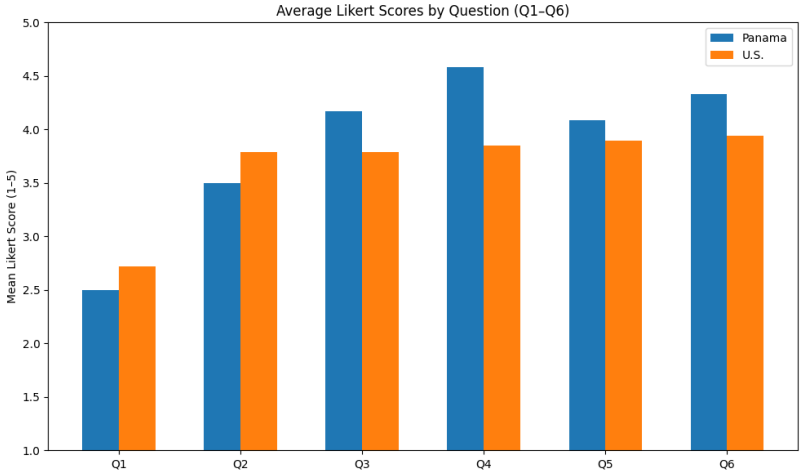

Figure 1: Average Likert scores (Q1–Q6) for U.S. and Panama students.

- **Q4 and Q5: Future willingness.** Panama students expressed the strongest enthusiasm for future international programs (Q4 mean 4.58) and future cross-border courses (Q5 mean 4.33). U.S. students were also positive (means of 4.04 and 4.14), but slightly less. High Panama enthusiasm persisted despite frustrations with uneven participation.
- **Q6: Platforms.** Both groups rated the platforms as supportive, with Panama students reporting a slightly higher mean (4.20) than U.S. students (4.00). The small difference indicates that technical tools were not major barriers compared to coordination and accountability challenges.

3.2 Qualitative Comparison (Q7)

Figure 2 shows theme frequencies from open-ended reflections. It is important to note that U.S. students submitted nearly four times as many reflections as Panama students, which naturally increases the absolute number of theme mentions.

Panama reflections, while fewer, were more pointed and focused on specific collaboration barriers, whereas U.S. reflections tended to be longer, more expressive, and more varied in tone. U.S. students frequently described uneven workload distribution, teammate disengagement, and late-stage integration failures. Panama students consistently emphasized challenges related to communication, scheduling mismatches, and delayed backend availability.

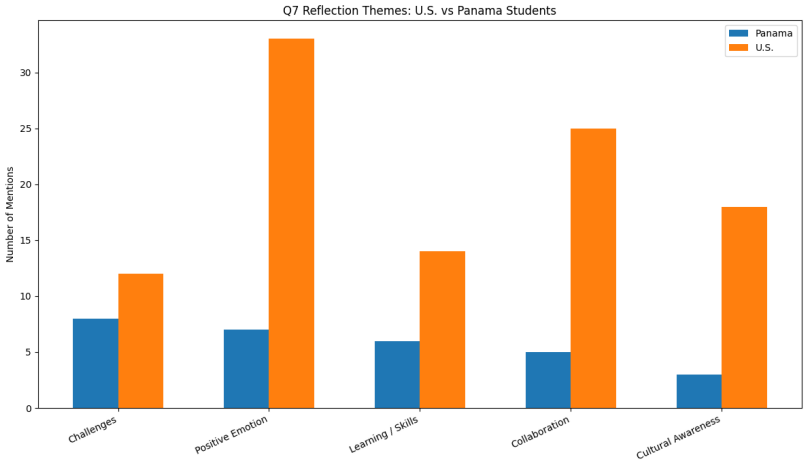

Figure 2: Comparison of Q7 reflection themes between U.S. and Panama students.

While both groups reported positive emotional experiences and appreciation of cross-cultural collaboration, Panama students demonstrated more consistent motivation and stronger desire for deeper interaction. Several Panama reflections noted that they "waited for backend updates that never arrived" or had to "complete tasks with little response from the U.S. side." Only four of the eight teams achieved full integration, and in every unsuccessful case, backend delays were the documented bottleneck.

4 Instructor Reflections

The instructor perspectives provide additional clarity on the asymmetries observed in the student data. On the Panama side, the collaborating instructor expressed enthusiasm for the partnership but was unable to remain consistently involved due to illness and scheduling constraints. As a result, only a quarter of Panama students participated in the survey. Several of their reflections indicate that they assumed a higher degree of reliability and technical readiness from their U.S. partners than they ultimately encountered.

From the U.S. instructional perspective, the six-week overlap occurred exclusively during the coding phase of the U.S. course, which created a favorable alignment of technical tasks but also magnified differences in student preparation and accountability. The collaboration exposed structural weaknesses common in broad-access computing programs. Many students entered the

course with uneven preparation in programming, limited teamwork experience, and inconsistent habits of communication. Several teams struggled to complete required backend components before integration week, and in cases where integration failed entirely, the bottleneck was always on the U.S. side. These patterns were visible not only in survey scores but in the weekly reports and reflections describing unresponsive teammates, skipped meetings, and last-minute code changes.

Rather than viewing COIL as the cause of these challenges, both instructors recognized that the international dimension intensified pre-existing disparities in preparation, expectations, and accountability. The module ultimately functioned only because instructors established strict milestones, monitored weekly progress, and intervened when teams stalled. These observations reinforce that successful COIL implementation requires strong local scaffolding, structured communication protocols and consistent supervision.

5 Discussion and Conclusion

The combined data illustrate that COIL does not neutralize or compensate for inconsistent preparation, uneven motivation, or weak teamwork skills. Instead, it magnifies these issues. Students at the Panama institution approached the project with high motivation, steady work habits, and strong expectations of accountability. Many U.S. students, however, demonstrated irregular participation, weak foundational skills, and intermittent communication. The resulting imbalance directly shaped the collaboration experience and contributed to Panama students' frustration.

These findings highlight the need for strong local scaffolding before students engage in cross-cultural work. In broad-access computing programs, COIL should be implemented only when students demonstrate baseline teamwork competencies and reliable follow-through. Without enforced milestones, explicit communication rules, and consistent instructor oversight, international collaboration may become one-sided and place undue burden on more motivated teams.

If we were to implement this COIL module again, several adjustments would be necessary to improve collaboration outcomes. First, clearer and more frequent milestones would be established earlier in the project to reduce last-minute integration failures. Second, integration activities would be introduced sooner, rather than being concentrated near the end of the module. Third, team sizes would be reduced to increase individual accountability and visibility of participation. In addition, mandatory weekly check-ins with clearly defined deliverables would be enforced to ensure consistent engagement across institutions. Finally, expectations for communication frequency, response times, and

responsibility for integration tasks would be made explicit at the outset of the collaboration.

Despite the challenges, both groups expressed meaningful learning outcomes. Panama students maintained exceptionally strong enthusiasm for future COIL activities, and many U.S. students reported their first authentic experience with distributed development. When paired with well-aligned institutional structures and adequate preparation, COIL remains a powerful model for teaching real-world collaboration in computing education.

References

[1] Darla K. Deardorff. "Identification and Assessment of Intercultural Competence as a Student Outcome of Internationalization". In: *Journal of Studies in International Education* 10.3 (2006), pp. 241–266. DOI: 10.1177/1028315306287002. URL: https://journals.sagepub.com/doi/10.1177/1028315306287002.

[2] Olivera Grujic. "Integrating COIL into an Undergraduate Software Engineering Course: A Cross-Cultural Experience Report". In: *Journal of Computing Sciences in Colleges* 41.8 (2026).

[3] Robert O'Dowd. "From Telecollaboration to Virtual Exchange: State-of-the-Art and the Role of UNICollaboration in Moving Forward". In: *Journal of Virtual Exchange* 1 (2018), pp. 1–23. DOI: 10.14705/rpnet.2018.jve.1. URL: https://doi.org/10.14705/rpnet.2018.jve.1.

Sonifying Kepler's Harmonice Mundi: A Csound Implementation for Teaching Scientific Sonification*

Paul R. Viotti[1], Abbas Attarwala[2], Hrishikesh Vasant Hasabnis[2]
[1]Political Science Department
California State University, Chico
Chico, CA 95973, USA
pviotti@csuchico.edu
[2]Computer Science Department
California State University, Chico
Chico, CA 95973, USA
{aattarwala,hvhasabnis}@csuchico.edu

Abstract

This article presents a computational approach to sonifying Johannes Kepler's "music of the spheres" as described in Book V of his *Harmonice Mundi* (1619). Using Csound, we map planetary angular velocities to audio frequencies, creating an audible representation of the solar system's orbital mechanics. The original composition, rendered in stereo sound, positions the listener at the Sun to experience the planets in their orbits. The implementation demonstrates core concepts in scientific sonification, data-driven sound synthesis, and interdisciplinary computing. We provide annotated code examples suitable for classroom use in computer science, digital audio, or computational science courses. The project illustrates how historical scientific ideas can motivate engaging programming exercises while teaching fundamental concepts in signal processing and data transformation.

1 Introduction

In 1619, Johannes Kepler published *Harmonice Mundi* (*The Harmony of the World*), in which he proposed that the planets produce a kind of celestial music through their orbital motions. Kepler calculated the angular velocities of each planet at its closest approach to the Sun (perihelion) and its farthest point (aphelion), then mapped these velocities to musical pitches. Kepler's music was metaphorical, as he understood that no literal sound propagates through space. Yet the mathematical framework he developed provides an elegant foundation for the sonification of astronomical data.

Sonification, the systematic mapping of data to sound, has become an important technique in scientific data representation, accessibility, and exploratory data analysis [6, 4]. The field has grown rapidly in recent years: [13] identified nearly 100 sound-based astronomy projects alone, while NASA's Chandra X-ray Observatory sonification program has demonstrated broad public engagement and measurable learning gains among both sighted and blind or low-vision participants [1]. Sonification is now recognized as a tool not only for scientific discovery but also for science communication and accessibility across disciplines [9, 3]. As [12] note, auditory displays are particularly effective when information has complex temporal patterns or when the visual system is occupied with other tasks. From monitoring network traffic to representing stock market fluctuations, sonification transforms abstract numerical relationships into audible patterns that humans can perceive intuitively. Kepler's planetary music offers an ideal pedagogical entry point: the underlying physics is well understood, the data is publicly available, and the result is immediately compelling to listeners.

This article describes our implementation of Kepler's planetary music using Csound, a widely used audio-programming language developed at MIT. The project exemplifies approaches to computer music synthesis documented in [8], while applying them to scientific data. We present the complete workflow from astronomical data to rendered audio, with code examples suitable for use in computer science education. The project integrates concepts from orbital mechanics, signal processing, and creative coding, making it appropriate for interdisciplinary courses or as a capstone exercise in digital audio programming.

2 Kepler's Sonification Model

Kepler's approach to planetary music rests on a simple principle: pitch is proportional to angular velocity [10]. As a planet orbits the Sun, its speed varies according to Kepler's Second Law (the law of equal areas). At perihelion, when the planet is closest to the Sun, it moves fastest; at aphelion, when farthest, it

moves slowest. This variation in velocity maps directly to a variation in pitch.

2.1 Velocity-to-Pitch Mapping

For each planet, we compute the angular velocity (rate of change of orbital position as seen from the Sun) at regular time intervals. These velocities are then linearly scaled to frequencies in the audible range. Saturn, the slowest and outermost planet known to Kepler, produces the lowest pitches, while Mercury, the fastest, produces the highest. The scaling can be adjusted to place all voices in a comfortable listening range.

The pitch at any moment is determined by:

$$\text{frequency}(t) = \text{scale\_factor} \times \text{angular\_velocity}(t) \tag{1}$$

where the scale factor is chosen to place Saturn's aphelion velocity at the low end of the desired pitch range. This preserves Kepler's original intent—pitch proportional to angular velocity—while ensuring all planetary voices fall within a comfortable listening range.

2.2 Pitch Ratios and Musical Intervals

Kepler was particularly interested in the ratio between each planet's highest and lowest pitch, which depends on orbital eccentricity. For a planet with eccentricity e, the velocity ratio (and hence pitch ratio) between perihelion and aphelion is given by:

$$\text{pitch\_ratio} = \frac{1+e}{1-e} \tag{2}$$

Mercury, with the highest eccentricity ($e \approx 0.206$), spans nearly a musical fifth. Earth and Venus, with nearly circular orbits, produce only slight pitch variation. These ratios approximate simple harmonic intervals, a coincidence that Kepler interpreted, from his 16th/early-17th century ontology, as evidence of cosmic design.

3 Implementation in Csound

Csound is a domain-specific language for audio synthesis, originally developed by Barry Vercoe at the MIT Media Lab in 1985 [11, 2]. It belongs to the Music N family of languages that trace back to Max Mathews's pioneering work at Bell Labs [8]. A Csound composition consists of two components: an orchestra file (`.orc`) defining instruments, and a score file (`.sco`) specifying musical events. This separation of timbre from structure maps naturally to our sonification task.

The lineage of computer music languages begins with Mathews's MUSIC I (1957), the first program to generate digital audio waveforms. Subsequent versions—MUSIC II through MUSIC V—refined the paradigm of unit generators: modular signal-processing components that could be combined to create complex sounds. Vercoe's earlier Music 11 (1973) introduced the distinction between audio-rate and control-rate signals, an optimization that Csound inherited and extended. Written in portable C, Csound could run on virtually any computing platform, democratizing access to sophisticated synthesis techniques previously confined to expensive mainframes and specialized hardware. Nearly four decades later, Csound remains actively maintained and widely used in both academic research and professional production.

3.1 Data Preparation

We computed orbital positions and angular velocities for each planet (Mercury through Saturn) spanning one complete Saturn orbit—approximately 29.5 Earth years [7]. The resulting dataset contains roughly 21,500 time steps per planet. For Mercury, a representative data excerpt appears below:

```
TIME     xs        ys          frequency
0        1.2059    0           168.85
0.5      1.20453   0.0530412   168.906
1        1.20044   0.10591     169.074
...
44       -0.7941   -0.000713   389.38    ; perihelion (max freq)
...
88       1.20589   0.00329     168.85    ; one orbit complete
```

The columns represent: time in days, x-coordinate, y-coordinate, and instantaneous frequency. The frequency column shows the relative pitch value derived from angular velocity, which is lowest at aphelion and highest at perihelion. Mercury completes one orbit every 88 Earth days. The x and y coordinates describe the planet's position in the orbital plane, used for spatial positioning in the stereo field.

3.2 The Score File

The Csound score file translates our data into note events. Each planet is assigned an instrument number, and its trajectory is encoded as a series of pitch points that Csound interpolates as a continuous glissando. We compress the full Saturn year into 120 seconds of audio:

```
; kepler.sco - Score file for planetary sonification
; Time compression: 1 Saturn year = 120 seconds
```

```
; Each 'i' statement: instrument, start, duration, freq, pan

; Mercury (instrument 1) - excerpt showing glissando points
i1  0.000  0.005  168.85   0.50   ; t=0, centered
i1  0.005  0.005  168.91   0.52   ; slight pan right
i1  0.010  0.005  169.07   0.54
i1  0.015  0.005  169.35   0.56
; ... continues for ~25,000 events per planet

; Saturn (instrument 6) - much slower variation
i6  0.000  120.0  20.15    0.50   ; single long tone

; Note: In practice, we use GEN routines to load frequency
; tables rather than explicit note events for efficiency.
```

The time compression ratio is approximately 10,759 Earth days compressed into 120 seconds, meaning each second of audio represents about 90 Earth days of orbital motion.

3.3 The Orchestra File

The orchestra file defines the sound-generating instruments. Each planet is rendered as a sine wave oscillator with its frequency driven by the score data. We add two enhancements: stereo panning based on orbital position, and a subtle tremolo effect tied to each planet's axial rotation period.

```
; kepler.orc - Orchestra file for planetary sonification

sr = 44100      ; sample rate
kr = 4410       ; control rate
ksmps = 10      ; samples per control period
nchnls = 4      ; quadraphonic output (use 2 for stereo)

; Instrument 1: Mercury
; p4 = frequency, p5 = pan angle (0-360 degrees)
instr 1
  ifreq = p4                          ; base frequency
  ipan = p5                           ; orbital position

  ; Tremolo based on Mercury's rotation (58.6 Earth days)
  ; Scaled to perceptible rate in compressed time
  ktrem lfo 0.08, 3.2, 1              ; depth, rate, sine

  ; Main oscillator with tremolo modulation
  asig oscil (0.7 + ktrem), ifreq, 1  ; amp, freq, table
```

```
  ; Quadraphonic panning (or stereo with locsig)
  a1, a2, a3, a4 locsig asig, ipan, 1, 0.1

  outq a1, a2, a3, a4
endin

; Instruments 2-6 follow same pattern with planet-specific
; tremolo rates derived from axial rotation periods:
;   Venus:   4.1 Hz  (retrograde, 243 days)
;   Earth:   5.8 Hz  (24 hours)
;   Mars:    5.6 Hz  (24.6 hours)
;   Jupiter: 8.4 Hz  (9.9 hours)
;   Saturn:  8.0 Hz  (10.7 hours)
```

The tremolo adds textural differentiation between planets while encoding an additional astronomical parameter—each planet's rotation on its own axis. This layering of data dimensions (orbital velocity → pitch, orbital position → pan, axial rotation → tremolo) demonstrates a key principle in sonification design: independent parameters can be mapped to perceptually distinct audio attributes.

3.4 Rendering and Output

Csound renders the composition to a multi-channel audio file. The original version was produced in quadraphonic surround sound, with each planet positioned spatially in the four-channel field according to its orbital location. A stereo mixdown condenses this spatial information into left-right panning. The rendering command is straightforward:

```
csound -o kepler.aiff kepler.orc kepler.sco
```

The resulting audio presents all six planets simultaneously, each warbling through its pitch range at its characteristic orbital period. Mercury's rapid oscillations contrast with Saturn's slow, bass drone, creating a surprisingly musical texture from purely astronomical data.

4 Spatial Audio Design

The live presentations described the intended perspective: "Imagine that you are sitting on the sun." From this heliocentric vantage point, planets orbit around the listener, their positions in the sound field corresponding to their orbital coordinates.

The stereo implementation preserves this spatial concept through left-right

panning, although a few modifications to the Csound orchestra file can render the piece in 3D, quadraphonic sound. In the current version, each planet's x-coordinate from the orbital data determines its position in the stereo field. As Mercury races through its orbit, its voice sweeps rapidly from left to right and back. This simple mapping transforms abstract positional data into intuitive spatial motion that listeners can track even without visual reference.

The panning adds a second layer of information to the sonification: pitch encodes velocity while stereo position encodes orbital location. Listeners can thus perceive both how fast a planet is moving (pitch) and where it is in its orbit (left-right position). This dual encoding exemplifies how sonification can convey multiple data dimensions simultaneously—an advantage of auditory display noted in the sonification literature [4].

5 Pedagogical Applications

The composition described in this article has been presented publicly as a quadraphonic installation at the San Francisco Art Institute (2003) and as a lecture demonstration at an international conference on Kepler's legacy in St. Petersburg, Russia (2019). Neither presentation involved a written paper or peer-reviewed proceedings; the present article represents the first formal publication of the implementation and its pedagogical framework. Although the project has not yet been deployed in a formal course setting, it is designed to teach concepts spanning computer science, digital audio, and computational science. The layered encoding of astronomical parameters—orbital velocity mapped to pitch, orbital position to spatial location, axial rotation to tremolo—exemplifies the parameter mapping approach to sonification design [12]. Below we outline intended learning outcomes and suggest exercises for adapting the material to various courses.

5.1 Learning Outcomes

- **Data transformation:** Students learn to map numerical data from one domain (orbital mechanics) to another (audio frequencies), a fundamental operation in scientific computing and visualization.
- **Signal processing fundamentals:** The project introduces oscillators, amplitude modulation (tremolo), and stereo panning—core concepts in digital signal processing.
- **Domain-specific languages:** Csound exemplifies a DSL optimized for a particular problem domain, illustrating language design tradeoffs between expressiveness and generality.

- **Interdisciplinary computing:** Students engage with historical primary sources (Kepler's original text) and astronomical data, demonstrating how computing intersects with other disciplines.
- **Creative coding:** The project encourages experimentation—students can modify timbres, add effects, or sonify different datasets using the same framework.

5.2 Suggested Exercises

1. **Extend to outer planets:** Add Uranus and Neptune (discovered after Kepler) to the composition. How do their long orbital periods affect the sonic result?
2. **Alternative timbres:** Replace sine waves with more complex waveforms (sawtooth, square) or sampled instruments. How does timbre affect the perception of planetary motion?
3. **Tempo variation:** Experiment with different time compression ratios. What is the effect of a 60-second piece versus a 10-minute piece?
4. **Sonify different data:** Apply the same mapping technique to stock prices, climate data, or network traffic. What makes some datasets more amenable to sonification than others?
5. **Port to another language:** Reimplement the sonification in Python (using pyo or SuperCollider bindings), JavaScript (using Web Audio API), or another audio environment. Compare the development experience.

5.3 Sample Assignment Prompt

The following assignment prompt is offered for instructors who wish to adopt this project directly. It assumes students have introductory programming experience and access to a Csound installation.

Assignment: Planetary Sonification. Using the Csound orchestra and score files provided in this article as a starting point, complete the following tasks: (1) Modify the frequency scaling so that Earth's aphelion pitch is concert A (440 Hz). Recompute the scale factor and document your calculation. (2) Add a seventh instrument for the Moon, using the Moon's orbital period around Earth (approximately 27.3 days) and its orbital eccentricity ($e \approx 0.055$) to determine its pitch range. (3) Replace the sine wave oscillator for one planet with a more complex waveform of your choice and describe, in one paragraph, how the timbral change affects the listener's perception of that planet's data.

Submit your modified `.orc` and `.sco` files, a rendered audio file, and a brief written reflection (300–500 words) comparing your sonification choices to the original design.

6 Conclusion

Kepler envisioned the planetary orbits as an "everlasting polyphony"—a cosmic music inaudible to human ears but mathematically precise [5]. Our Csound implementation renders this polyphony audible, transforming abstract orbital data into a layered composition where each planet's voice reflects its astronomical character. Mercury chatters rapidly through its eccentric orbit while Saturn drones beneath, and the listener, stationed imaginatively on the Sun, hears the solar system sing.

Beyond its aesthetic appeal, the project demonstrates principles central to computer science education: data transformation, domain-specific languages, and the creative application of programming to interdisciplinary problems. While the specifics here involve Kepler and Csound, the underlying pedagogical approach is transferable. Any dataset with meaningful temporal or multivariate structure—climate records, seismic traces, genomic sequences, financial time series—can be sonified using the same parameter-mapping framework. The essential skills students practice—selecting data dimensions, choosing perceptually appropriate audio mappings, implementing the synthesis pipeline, and critically evaluating the result—are applicable well beyond this particular project. Recent empirical work supports the value of such approaches: [1] found significant self-reported learning gains among participants who engaged with sonified astronomical data, and [13] documented the rapid growth of sound-based astronomy projects aimed at education and public engagement.

The Csound code is intentionally accessible, suitable for students with introductory programming experience, and is included in full in this article. We encourage educators to adapt and extend this material for their own courses. The complete sonification will be performed live at the conference presentation. We hope this project inspires both scientific curiosity and creative experimentation.

A Companion Audio Resources

To support classroom use and independent exploration, audio recordings of the sonification are available as downloadable files. Listeners can audition each planet individually to hear its characteristic pitch range and tremolo, then experience the full six-voice texture in the combined mix.

Individual Planet Tracks (10 seconds each):

- Mercury (fastest orbit, highest pitch, widest pitch range)
- Venus (nearly circular orbit, minimal pitch range)
- Earth (very low eccentricity, minimal pitch range)
- Mars (moderate eccentricity)
- Jupiter (slow orbital period, low pitch)
- Saturn (slowest orbit, lowest pitch)

Combined Mix:
- All Six Planets (full composition, one Saturn year)

We recommend using headphones or quality speakers to appreciate the full frequency range. The spatial positioning is best experienced in a quiet listening environment. Audio files and additional materials are available from the corresponding author upon request.

Acknowledgments

This work builds upon the original sonification created by the first author in collaboration with the late Ralph Herman Abraham (1936–2024), Professor Emeritus of Mathematics at the University of California, Santa Cruz. Professor Abraham was a pioneering figure in dynamical systems theory, visual mathematics, and the cultural dimensions of scientific thought. He conceived the Visual Kepler project and provided the mathematical framework for mapping Kepler's planetary velocities to audible frequencies. The original composition was first performed in quadraphonic surround sound at the San Francisco Art Institute in April 2003. It was later presented as a lecture demonstration at "The Harmony of the World," an international conference celebrating the 400th anniversary of Kepler's *Harmonices Mundi*, hosted by the Sophia Centre for the Study of Cosmology in Culture at the University of Wales Trinity Saint David. The conference was held at the State Museum of the History of Religion in St. Petersburg, Russia, June 21–23, 2019, and included an excursion to view Kepler's original manuscripts at the Archive of the Russian Academy of Sciences. We are grateful for Professor Abraham's vision, mentorship, and enduring contributions to interdisciplinary scholarship. This article is dedicated to his memory. Finally, we gratefully acknowledge the use of Claude for proofreading, grammatical checks, and other text editing tasks.

References

[1] Kimberly K. Arcand et al. "A Universe of Sound: Processing NASA Data into Sonifications to Explore Participant Response". In: *Frontiers*

in Communication 9 (2024), p. 1288896. DOI: 10.3389/fcomm.2024.1288896.

[2] Richard Boulanger. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. MIT Press, 2000.

[3] Rhea Braun, Maxwell Tfirn, and Roseanne M. Ford. "Listening to Life: Sonification for Enhancing Discovery in Biological Research". In: *Biotechnology and Bioengineering* 121.10 (2024), pp. 3009–3019. DOI: 10.1002/bit.28729.

[4] Thomas Hermann, Andy Hunt, and John G. Neuhoff, eds. *The Sonification Handbook*. Logos Verlag Berlin, 2011.

[5] Johannes Kepler. *The Harmony of the World*. Trans. by E. J. Aiton, A. M. Duncan, and J. V. Field. Original work published 1619. American Philosophical Society, 1997.

[6] Gregory Kramer, ed. *Auditory Display: Sonification, Audification, and Auditory Interfaces*. Addison-Wesley, 1994.

[7] Patrick Moore et al. *The Atlas of the Solar System*. Rev. Crescent Books, 1990.

[8] Curtis Roads. *The Computer Music Tutorial*. MIT Press, 1996.

[9] Matt Russo et al. "Improving Earth Science Communication and Accessibility with Data Sonification". In: *Nature Reviews Earth & Environment* 5.1 (2024), pp. 1–3. DOI: 10.1038/s43017-023-00512-y.

[10] Bruce Stephenson. *The Music of the Heavens: Kepler's Harmonic Astronomy*. Princeton University Press, 1994.

[11] Barry L. Vercoe and Dan P. W. Ellis. "Real-time Csound: Software Synthesis with Sensing and Control". In: *Proceedings of the 1990 International Computer Music Conference*. International Computer Music Association, 1990, pp. 209–211.

[12] Bruce N. Walker and Michael A. Nees. "Theory of Sonification". In: *The Sonification Handbook*. Ed. by Thomas Hermann, Andy Hunt, and John G. Neuhoff. Logos Verlag Berlin, 2011, pp. 9–39.

[13] Anita Zanella et al. "Sonification and Sound Design for Astronomy Research, Education and Public Engagement". In: *Nature Astronomy* 6 (2022), pp. 1241–1248. DOI: 10.1038/s41550-022-01721-z.

Enhancing Students' Understanding of SQL Aggregate Functions in Nested Subqueries *

Nifty Assignment

Jamil Saquer
Computer Science Department
Missouri State University, Springfield, MO 65897
jamilsaquer@missouristate.edu

SQL is a foundational skill taught in the introductory database course. Aggregate functions are commonly used to summarize data across groups of rows in a database. For instance, in a university database, they can be used to calculate the number of students in each department, and to determine the average number of credit hours taken by students in a given semester.

However, using aggregate functions, especially in subqueries, can be challenging due to specific rules governing their use. Students often struggle with these rules, as they are rarely discussed in detail in popular database textbooks. This proposal introduces an assignment designed to enhance students' understanding of aggregate functions, particularly their use in nested subqueries.

The assignment begins with a review of the following key points:

- SQL aggregate functions are typically used in the SELECT and HAVING clauses.
- Aggregate functions can appear in the WHERE clause if used within a subquery, provided that the subquery is fully computed and its result is available before the outer query executes.
- In many correlated subqueries, the inner query is computed for each tuple of the outer query. A correlated subquery is a nested query that references columns from the outer query and is evaluated once per tuple of the outer query.

The assignment is based on a database comprising the following relations, with primary key attributes underlined:

student(id, name, deptName, totalCredits)
takes(id, courseId, semester, year, grade)
course(courseId, title, deptName, credits)

Students are required to use a database management system such as Oracle or PostgreSQL to write SQL queries for the following tasks:

1. Determine the number of students in each department.
2. Identify the department name and the number of students for departments with at least three students.
3. Use the WITH clause to create a temporary relation that finds the names of students who have taken at least three courses.
4. Use the set membership operator to identify students who have taken at least three courses.
5. Extend the previous query to find students who have taken at least three courses specifically in the Fall 2020 semester.
6. Use the EXISTS construct to find students who have taken at least three courses.
7. Write a correlated subquery in the WHERE clause to find students who have taken at least three courses.
8. Use a derived relation (possibly with the LATERAL clause) to find students who have taken at least three courses.
9. Determine the department(s) that has offered the most number of courses.
10. Calculate the average number of courses taken by students during the Fall 2020 semester.

The questions in this assignment progress in difficulty to encourage students to solve all of them. The assignment is given after the main SQL material has been covered and it follows three other SQL homework assignments. The assignment is designed to reinforce key concepts, including aggregate functions, nested subqueries, and the rules for using aggregate functions within nested subqueries.

In prior semesters without this assignment, the author observed that some students struggled with writing SQL queries similar to what is needed to answer some of the questions in this exercise. However, fewer students faced such difficulties when the assignment was incorporated. Additionally, written comments on the course evaluations revealed that several students found this assignment helpful in improving their understanding of SQL, particularly as it required them to explore multiple methods for answering the same query, as demonstrated in questions 3 through 8.

Boosting Black-Box Software Testing with Structured Prompt Design for Generative AI *

Nifty Assignment

Rad Alrifai
Department of Mathematics and Computer Science
Northeastern State University
Tahlequah, OK 74464
alrifai@nsuok.edu

Software testing is an important topic in software engineering, where students learn essential black-box methods such as Equivalence Class Testing (ECT) and Boundary Value Analysis (BVA). As the use of AI becomes increasingly common in software development, there are increasing opportunities to utilize AI-supported testing into software engineering courses. This assignment integrates the use of generative AI with classical black-box testing by using a structured prompt-engineering. Students learn to design clear, goal-driven prompts to create black-box test cases for a campus-focused social media application.

The assignment is designed to provide students with hands-on experience in using AI-assisted test case generation. First, students manually identify equivalence classes and boundary values for different variables in an assigned case study. Second, the instructor provides guidance on responsible use of AI in testing, reviews sample prompts, shares recommended prompt formats, and helps students avoid common prompt writing errors. Next, students iteratively refine the prompts to instruct a generative AI tool to produce a set of test cases. Continuously, students compare the test cases produced by AI with the manually created test cases while evaluating the correctness, completeness, and consistency of applying ECT and BVA principles. This comparison helps students use AI more effectively to generate alternative edge cases or broadening test coverage, while validating the accuracy of AI generated-outputs.

Thus, this assignment provides students the necessary practice to apply prompt engineering in software testing while they continue to learn and apply essential software testing techniques. Integrating common software testing methods like ECT and BVA with AI tools helps students understand the practical capabilities and constraints of AI in modern software engineering practice.

Firmware Reverse Engineering with patch-hunter*

Nifty Assignment

Michael Ham
Beacom College of Computer and Cyber Sciences
Dakota State University
Madison, SD 57042
michael.ham@dsu.edu

Universities can attain a National Center of Academic Excellence in Cybersecurity (NCAE-C) designation by meeting the requirements developed by the National Security Agency (NSA). These designations support quality academic programs that contribute to the nation's cyber workforce in the most critical areas. The Cyber Operations (CAE-CO) designation emphasizes advanced technical skill sets related to exploitation, reverse engineering, and computer science.

CAE-CO degree programs are evaluated against the agency's prescribed Knowledge Units (KUs) that drive curricular development and learning outcomes. Several KUs require hands-on labs and applied work, offering insight into intelligence community priorities. Software Reverse Engineering is a mandatory KU, while Hardware Reverse Engineering and Microcontrollers are listed as optional KUs. Together, these KUs highlight the importance of analyzing the security of embedded systems, including wireless routers, IoT, critical infrastructure, and operational technology (OT). Notable IoT compromises, such as those enabling the Mirai botnet further underscore the importance of learning capabilities in this area. To support these learning outcomes, this paper presents a nifty assignment built around patch-hunter, an author-developed Docker-based firmware analysis toolkit, used to teach firmware differential analysis. The assignment is designed for upper-level undergraduate students with a strong background in computer science and cybersecurity and integrates into existing reverse engineering or software security courses without requiring dedicated embedded hardware labs.

patch-hunter is designed to impart knowledge about the process of firmware differential analysis, a crucial skill in reverse engineering and vulnerability research of embedded systems. In this experiential assignment, students use *patch-hunter* to analyze how firmware updates or patches that exist within a target wireless router firmware image can reveal newly addressed vulnerabilities, added features, regressions, or configuration modifications that may introduce vectors for compromise. The assignment focuses on Linux-based embedded firmware, which dominates consumer and enterprise networking devices, allowing students to engage with realistic and widely deployed systems.

Students can build the *patch-hunter* Docker container locally or run it from an online container registry. Using *patch-hunter*, students extract Linux-based embedded file systems from two versions of a target router's firmware, compare their internal structures, and generate structured JSON outputs that elucidate modified, added, or removed files. This process of inventorying firmware changes mirrors professional vulnerability analysis workflows as it helps analysts minimize the data set for subsequent in-depth analysis. Although the assignment does not require physical device teardown, it introduces hardware reverse engineering concepts by analyzing firmware extracted from embedded devices, examining initialization scripts, configuration files, and binaries that directly interface with hardware components.

The assignment guides students through querying the JSON results using jq, with a focus on web application files and other high-value targets that have historically led to compromise, such as modified CGI scripts or updated binaries related to router authentication. Once a suspect file is identified, students perform a unified diff or utilize Ghidra's Version Tracking to inspect it for vulnerabilities. This exercise allows students to demonstrate technical proficiency across multiple tools including Docker, Binwalk, jq, and Ghidra while reinforcing fundamental concepts in software reverse engineering, binary analysis, and vulnerability assessment. This process reinforces reverse engineering as the reconstruction of program behavior and intent from compiled artifacts.

Upon completion, students are able to extract and compare embedded firmware images, identify security-relevant changes introduced by patches, and apply reverse engineering techniques to assess potential vulnerabilities through screenshots and written analysis of key workflow stages. To achieve higher levels of cognitive activity, students also provide written evaluations of their findings to synthesize results in the context of firmware reverse engineering, helping bridge tool usage with conceptual understanding and interpretation.

patch-hunter's containerized design ensures the assignment's portability, reproducibility, and suitability for both in-person and remote instruction. It is released as open source at `https://github.com/DSUmjham/patch-hunter`.

Theory and Practice in Functional Programming*

Nifty Assignment

Cong-Cong Xing[1], *Jun Huang*[2]
[1]*Department of Mathematics/Computer Science*
Nicholls State University
Thibodaux, LA 70310
cong-cong.xing@nicholls.edu
[2]*Dept. of Electrical Engineering and Computer Science*
South Dakota State University
Brookings, SD 57007
Jun.Huang@sdstate.edu

It is well-known that λ-calculus is the foundation of functional programming languages, and that a good understanding on λ-calculus can significantly help one feel comfortable in coding functional programs. Unfortunately, dealing with mathematical/theoretical materials is typically a daunting task for computer science majors, and λ-calculus is no exception. As such, carefully-crafted exercises in λ-calculus designed to enhance students' grasp on this subject are much needed.

The purpose of this nifty assignment is to just provide such an exercise. Suppose we are given the following syntax of applied λ-calculus (which is an extension of the pure λ-calculus)[1]

M ::= c | x | M1M2 | λx.M (c – constant)
c ::= tru | fls | if | 0 | iszero | pred | succ | fix

Students are asked to use this applied λ-calculus to do the following:

(1) Devise a function F (preferred as the fixed point of some term) that takes

[1]The reduction rules of the applied λ-calculus are omitted here to avoid potential clog of reading. But they are intuitively understandable. For example, pred(succ M) reduces to M, iszero 0 reduces tru, if tru M N reduces to M, and fix F reduces to F(fix F).

an integer parameter[2] n, a function parameter f, as well as an integer parameter x, and returns the result of applying f to x n times. For example, if $n = 3$, then the returned result would be $f(f(f\,x))$.

(2) Find out what $F\,2$ evaluates to (2 is an "abbreviation" of $succ(succ\,0)$) with all derivation steps.

(3) For any (function) g and (integer) y, show that $(F\,2)(F\,2)\,g\,y$ reduces to $g(g(g(g\,y)))$ (not $g(g(g\,y))$). (Hint: function application order.)

(4) Implement the devised function F in the actual functional language Standard ML (SML) and verify that the reduction of $(F\,2)(F\,2)\,g\,y$ in the previous step is correct by utilizing some specific g and y.

The solution to this assignment is given below with some elaborations.

(1) The initial solution of F should be something like

$$\mathsf{F} = \lambda\mathsf{n}.\lambda\mathsf{f}.\lambda\mathsf{x}.\mathsf{if}\ (\mathsf{iszero}\ \mathsf{n})\ \mathsf{x}\ (\mathsf{F}\ (\mathsf{pred}\ \mathsf{n})\ \mathsf{f}\ (\mathsf{f}\ \mathsf{x})),$$

and this recursive definition can be rewritten using the (intended) fixed point operator, fix, as

$$\mathsf{F} = \mathsf{fix}\ \mathfrak{F}$$

where

$$\mathfrak{F} = \lambda\mathsf{g}.\lambda\mathsf{n}.\lambda\mathsf{f}.\lambda\mathsf{x}.\mathsf{if}\ (\mathsf{iszero}\ \mathsf{n})\ \mathsf{x}\ (\mathsf{g}\ (\mathsf{pred}\ \mathsf{n})\ \mathsf{f}\ (\mathsf{f}\ \mathsf{x})).$$

(2) As such, we have

$$\begin{aligned}
&\mathsf{F}\ 2\\
=\ &(\mathsf{fix}\ \mathfrak{F})\ 2\\
=\ &\mathfrak{F}\ (\mathsf{fix}\ \mathfrak{F})\ 2\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathsf{if}\ (\mathsf{iszero}\ 2)\ \mathsf{x}\ ((\mathsf{fix}\ \mathfrak{F})\ (\mathsf{pred}\ 2)\ \mathsf{f}\ (\mathsf{f}\ \mathsf{x}))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathsf{if}\ (\mathsf{iszero}\ (\mathsf{succ}(\mathsf{succ}\ 0)))\ \mathsf{x}\ ((\mathsf{fix}\ \mathfrak{F})\ (\mathsf{pred}\ (\mathsf{succ}(\mathsf{succ}\ 0)))\ \mathsf{f}\ (\mathsf{f}\ \mathsf{x}))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ (\mathsf{fix}\ \mathfrak{F})\ (\mathsf{pred}\ (\mathsf{succ}(\mathsf{succ}\ 0)))\ \mathsf{f}\ (\mathsf{f}\ \mathsf{x})\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ (\mathsf{fix}\ \mathfrak{F})\ (\mathsf{succ}\ 0)\ \mathsf{f}\ (\mathsf{f}\ \mathsf{x})\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathfrak{F}\ (\mathsf{fix}\ \mathfrak{F})\ (\mathsf{succ}\ 0)\ \mathsf{f}\ (\mathsf{f}\ \mathsf{x})\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathsf{if}\ (\mathsf{iszero}\ (\mathsf{succ}\ 0))\ (\mathsf{f}\ \mathsf{x})\ ((\mathsf{fix}\ \mathfrak{F})\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{f}\ (\mathsf{f}(\mathsf{f}\ \mathsf{x})))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ (\mathsf{fix}\ \mathfrak{F})\ (\mathsf{pred}\ (\mathsf{succ}\ 0))\ \mathsf{f}\ (\mathsf{f}(\mathsf{f}\ \mathsf{x}))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ (\mathsf{fix}\ \mathfrak{F})\ 0\ \mathsf{f}\ (\mathsf{f}(\mathsf{f}\ \mathsf{x}))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathfrak{F}\ (\mathsf{fix}\ \mathfrak{F})\ 0\ \mathsf{f}\ (\mathsf{f}(\mathsf{f}\ \mathsf{x}))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathsf{if}\ (\mathsf{iszero}\ 0)\ (\mathsf{f}(\mathsf{f}\ \mathsf{x}))\ ((\mathsf{fix}\ \mathfrak{F})\ (\mathsf{pred}\ 0)\ \mathsf{f}\ (\mathsf{f}(\mathsf{f}(\mathsf{f}\ \mathsf{x}))))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathsf{if}\ \mathsf{tru}\ (\mathsf{f}(\mathsf{f}\ \mathsf{x}))\ ((\mathsf{fix}\ \mathfrak{F})\ (\mathsf{pred}\ 0)\ \mathsf{f}\ (\mathsf{f}(\mathsf{f}(\mathsf{f}\ \mathsf{x}))))\\
=\ &\lambda\mathsf{f}.\lambda\mathsf{x}.\ \mathsf{f}(\mathsf{f}\ \mathsf{x})
\end{aligned}$$

[2]Note that applied λ-calculus is untyped, and there is no way to specify the type of n to be integer in this framework. So the word "integer" (and others) here only indicates the intent.

So $F\,2$ ends up as a function that takes two parameters f and x (one at a time), and applies f to x twice.

(3) Let T denote $F\,2$ (for the convenience of writing). Then,

$$\begin{aligned}
\mathsf{(F\ 2)\ (F\ 2)\ g\ y} &= \mathsf{T\ T\ g\ y} \\
&= \mathsf{((T\ T)\ g)\ y} \\
&= \mathsf{(((\lambda f.\lambda x.\ f(f\ x))\ T)\ g)\ y} \\
&= \mathsf{T\ (T\ g)\ y} \\
&= \mathsf{(\lambda f.\lambda x.\ f(f\ x))\ (T\ g)\ y} \\
&= \mathsf{(T\ g)\ ((T\ g)\ y)} \\
&= \mathsf{(T\ g)\ (g(g\ y))} \\
&= \mathsf{g(g(g\ y)))}
\end{aligned}$$

The critical part of this derivation is that *function applications associate to left*. That is, PQR is a "shorthand" writing for $(PQ)R$ for any λ-terms P, Q, and R. If this evaluation rule is overlooked, a common miscalculation like the following may occur.

$$\begin{aligned}
\mathsf{(F\ 2)\ (F\ 2)\ g\ y} &\overset{1}{=} \mathsf{T\ T\ g\ y} \\
&\overset{2}{=} \mathsf{T\ (T\ g\ y)} \\
&\overset{3}{=} \mathsf{T\ (g\ (g\ y))} \\
&\overset{4}{=} \mathsf{g(g(g\ y))}
\end{aligned}$$

Here, two crucial errors take place. Step 2 is wrong because it is a violation of the left-associativity of function applications. Step 4 is wrong because it erroneously regards $T\,(g\,(g\,y))$ at step 3 as T taking two arguments g and $g\,y$, and carries out the evaluation that way. The correct understanding for $T\,(g\,(g\,y))$ is that T takes $g\,(g\,y)$ as one argument.

(4) The screenshot of SML/NJ implementation of F and the execution of $(F\,2)(F\,2)\,g\,y$ with $g(x) = x + 1$ and $y = 0$ is shown below.

```
Standard ML of New Jersey (32-bit) v110.99.5 [built: Mon Mar 18 15:01:51 2024]
-
- val rec F = fn n => (fn f => (fn x => if n=0 then x else F (n-1) f (f x) ));
val F = fn : int -> ('a -> 'a) -> 'a -> 'a
-
- val g = fn x => x+1;
val g = fn : int -> int
-
- val y = 0;
val y = 0 : int
-
- (F 2) (F 2) g y;
val it = 4 : int
```

We can see that the SML code for F is basically the same as the (recursively defined) λ-expression of F, which shows the importance of studying

theory. Also, the fact that $(F\,2)(F\,2)\,g\,y$ evaluates to 4 clearly verifies that $(F\,2)(F\,2)\,g\,y$ reduces to $g(g(g(g\,y)))$, as expected.

This assignment was given to a senior undergraduate functional programming class and followed by an in-class discussion. Students had already done some basic λ-calculus and SML practices when the assignment was assigned. The primary response from the class was that this assignment has combined and clarified many things, especially the $g(g(g(g\,y)))$ vs. $g(g(g\,y))$ issue about which most students are confused. We believe that this assignment is nifty since it conglomerates multiple important aspects of functional programming into one place and can be expected to produce the following gains.

- It offers a chance for students to see/practice how multi-variable functions (which are common in math) can be coded (or "curried") as single-variable but higher-order functions in λ-calculus which inherently requires all its functions to have only one variable.
- Considering that most students would have coded the function F by using loops if they were asked to implement it using some imperative language, this assignment demonstrates how loops (or the idea conveyed by loops) can be encoded as recursive functions in λ-calculus which does not have the loop construct.
- Along these lines, it must be realized that recursive function definitions rely on function *names*. But, function names are not an essential part of λ-calculus. In other words, functions are defined *directly* in λ-calculus without having to be named. Name tags of functions are something that we artificially introduce for pure convenience when working in λ-calculus. So a fundamental question is: how can we define recursive functions in λ-calculus (without using function names)? This assignment shows students that fixed point is the answer to the question.
- It fully anticipates the $g(g(g\,y))$ error that is due to the misordering of function applications, and prompts students to learn and adopt the critical "function applications associate to left" rule.
- The choice of the applied λ-calculus and the involvement of numbers in this assignment allow students to see how boolean constants and integers are handled at the theory level. But one thing that students must be aware of is that the 0 in applied λ-calculus is just a pure syntactic entity that resembles the the shape of number zero. It is *not* the number zero. So this assignment offers students a chance to discern between syntax and semantics of a language.
- The SML implementation portion allows students to see how theory and practice are connected in functional programming.

We hope that this assignment can be found useful by colleagues.

Ring a Doorbell: A Hands-On Introduction to Wireless Data Encoding*

Nifty Assignment

Kyle Cronin
Beacom College of Computer and Cyber Sciences
Dakota State University
Madison, SD 57042
kyle.cronin@dsu.edu

Cybersecurity education increasingly calls for activities that bridge abstract technical concepts with concrete, real-world systems. This assignment, "Ring a Doorbell," introduces students to the fundamentals of wireless data encoding through an interactive, accessible exploration of common household technology. Designed for use in secondary, postsecondary, and educator-professional-development settings, the activity leverages Software Defined Radios (SDRs), Raspberry Pis, and the increasingly familiar Flipper Zero to demystify wireless communication while reinforcing ethical cybersecurity practices.

Our university is one of the only in the region focused on developing scalable, hands-on cybersecurity learning experiences, with faculty creating a robust collection of labs and activities for learners ranging from novices to experts. This assignment extends that tradition by guiding students through the process of capturing and interpreting wireless signals generated by everyday devices—such as doorbell buttons, car key fobs, or remote controls—and connecting those observations to core concepts in data representation, radio frequency (RF) communication, and device security.

The activity begins with a short preparatory component in which students research basic RF terminology and consider where low-power wireless devices appear in daily life. This primes them to enter the lab session with context for the signal structures they will later observe. During the guided lab, students use SDRs to tune into the radio frequency used by a simple transmitter and collect waveform data produced during repeated button presses. Using visualizations generated by SDR software, students identify repeating patterns, mark

bit sequences, and interpret where and how a device encodes its unique identifier. Students often describe this moment as unexpectedly eye-opening—seeing the literal "1s and 0s" traveling through the air makes an otherwise invisible process suddenly tangible.

Once students establish a foundational understanding of wireless encoding, the assignment introduces two tools that are increasingly present—yet frequently misunderstood—in K–12 and college environments: the Flipper Zero and the Raspberry Pi. Rather than treating these devices as mysterious hacking gadgets, students explore their legitimate instructional uses, including signal visualization, controlled signal replay with approved devices, and demonstrations of responsible cybersecurity behaviors. This portion of the activity emphasizes ethics, safe experimentation, and pedagogically sound ways to integrate these tools into instruction.

The activity's premise is that it can be adapted for a specific audience—middle school learners, high school students, community college cohorts, or in-service educators. The premise of the equipment is unique in that many educators (and students, for that matter) have tools like a Flipper Zero on hand, yet struggle to find any sort of academic utility that may be applied with the tools.

This brief lesson outlines how the core wireless-encoding activity can be scaffolded, simplified, or enriched based on learners' prior knowledge and learning goals. This final step asks students not only to understand the technical concepts, but also to think like instructors: How does one translate an advanced cyber concept into an inclusive, developmentally appropriate learning experience?

Across implementations, the assignment consistently leads to strong engagement and reflective learning. Students report confidence with RF concepts, recognizing the educational potential of emerging cybersecurity tools, and appreciating the importance of ethical practice. Instructors benefit from a flexible activity that can scale from introductory computing courses to specialized cybersecurity workshops. Because the assignment makes use of inexpensive hardware and open-source software, it is easy to adopt or adapt at institutions of varying sizes and resources.

Ultimately, "Ring a Doorbell" provides a memorable, hands-on approach to teaching wireless data encoding—one that merges technical fluency with curiosity, analysis, and responsible innovation. It brings to life a part of computing that is often invisible and invites learners to explore how everyday technologies communicate, how signals can be studied, and how educators can use these tools to create meaningful cybersecurity learning experiences

Building a Linux Device Driver *

Nifty Assignment

Bin Peng
Computer Science and Information Systems
Park University
Parkville, MO 64152
bpeng@park.edu

Assignment Description

This assignment is for the input/output module in an undergraduate introductory operating system course. Students are required to build a read-only character device driver in C, install it on a virtual Linux machine, and unload it afterward. The purpose of the assignment is to practice building simple device drivers for UNIX like systems. Student comments indicated that many enjoyed seeing a device driver in action besides reading about it in the textbook.

This read-only character device driver tracks the number of times the device has been opened before the current request and prints out the count on user end given a read request. Upon a write request, however, the driver only generates a kernel message to log the request and return. Prior to this assignment, the course covered input/output module in operating systems and the role of device drivers in the I/O hierarchy on a typical computing system. Students have learned basic C syntax in a prerequisite course but do not otherwise have system programming experience. To prepare students for this assignment, two additional lectures, one on Linux Kernel module programming basics and the other on building Linux device drivers, are added to walk students through the steps. The assignment and lectures are mostly based on chapters 1 to 4 from the Linux Kernel Module Programming Guide [2]. Practices and the assignment are completed under the root account on a remote Linux machine at https://linuxzoo.net/ [1] via SSH.

The kernel module lecture covers the following from chapters 1, 2.1-2.2, and 2.5 of the Guide:

- The concept of kernel module,
- the `init_module()` and `cleanup_module()` functions,
- an example makefile to compile the code as a kernel module,
- and a walk-through of steps to load and unload the module and verify the results.

The Linux device driver lecture then explains additional topics needed from chapters 3 and 4 of the Guide:

- device file vs. device driver,
- the data structure in C to define a device for needed operations,
- C functions to add and remove a character device on Linux,
- an example device driver in C to replicate the `/dev/null` pseudo device, and
- the process to install the device driver, create a device file, and perform read and write on the device. Afterward, delete the device file and uninstall the driver.

The assignment then directs students to start with an existing character device file in ch4.1.5 chardev.c of the Guide. Students are asked to customize kernel messages, identify and fix a syntax error due to an outdated function call in `cleanup_module()`, and complete the whole process to install driver, create device file, read from/write to the device, and clean up. Figures 1 to 4 show the commands for this assignment.

```
[root@host-2-25 testchar]# make
make -C /lib/modules/3.10.0-514.10.2.el7.x86_64/build M=/root/testchar modules
make[1]: Entering directory `/usr/src/kernels/3.10.0-514.10.2.el7.x86_64'
  CC [M]  /root/testchar/chardev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/testchar/chardev.mod.o
  LD [M]  /root/testchar/chardev.ko
make[1]: Leaving directory `/usr/src/kernels/3.10.0-514.10.2.el7.x86_64'
[root@host-2-25 testchar]#
[root@host-2-25 testchar]# ls
chardev.c   chardev.mod.c  chardev.o  modules.order
chardev.ko  chardev.mod.o  Makefile   Module.symvers
```

Figure 1: The compilation and resulting files from the corrected code

```
[root@host-2-25 testchar]# insmod chardev.ko
[root@host-2-25 testchar]# lsmod
Module                  Size  Used by
chardev                12767  0
xt_CHECKSUM            12549  1
[root@host-2-25 testchar]# dmesg | tail -10
[  470.954143] chardev: loading out-of-tree module taints kernel.
[  470.954151] chardev: module license 'unspecified' taints kernel.
[  470.954154] Disabling lock debugging due to kernel taint
[  470.954227] chardev: module verification failed: signature and/or required ke
y missing - tainting kernel
[  470.954527] CS351char_YOUR_FULLNAME was assigned major number 248. To talk to
[  470.954529] the driver, create a dev file with
[  470.954531] 'mknod /dev/chardev c 248 0'.
[  470.954532] Try various minor numbers. Try to cat and echo to
[  470.954534] the device file.
[  470.954535] Remove the device file and module when done.
```

Figure 2: Load the device driver; use `lsmod` to verify the result and `dmesg` to view kernel messages.

```
[root@host-2-25 testchar]# mknod /dev/chardev c 248 0
[root@host-2-25 testchar]# ls -l /dev/chardev
crw-r--r--. 1 root root 248, 0 Dec 15 19:55 /dev/chardev
```

Figure 3: Create a new device file and verify

```
[root@host-2-25 testchar]# cat /dev/chardev
I already told you 0 times CS351char_YOUR_FULLNAME!
[root@host-2-25 testchar]# cat /dev/chardev
I already told you 1 times CS351char_YOUR_FULLNAME!
[root@host-2-25 testchar]# cat /dev/chardev
I already told you 2 times CS351char_YOUR_FULLNAME!
[root@host-2-25 testchar]# echo "some msgs" > /dev/chardev
-bash: echo: write error: Invalid argument
[root@host-2-25 testchar]# echo "a second try" > /dev/chardev
-bash: echo: write error: Invalid argument
[root@host-2-25 testchar]# dmesg | tail -5
[  470.954532] Try various minor numbers. Try to cat and echo to
[  470.954534] the device file.
[  470.954535] Remove the device file and module when done.
[  893.547660] Sorry, this operation isn't supported on CS351char_YOUR_FULLNAME.
[  917.383904] Sorry, this operation isn't supported on CS351char_YOUR_FULLNAME.
```

Figure 4: Perform read and write operations on the device; verify result of write through kernel messages

Additional Ideas

This assignment was written for students without system programming backgrounds. For a more advanced student group, the instructions may be simpli-

fied to let students fill in some of the code or steps. Students may also alter the given character device driver code to support more sophisticated read/write or additional operations.

References

[1] Gordon Russell. *Linuxzoo.net*. Accessed: 2025-12-15. 2025. URL: `https://linuxzoo.net/`.

[2] Peter Jay Salzman, Michael Burian, and Ori Pomerantz. *The Linux Kernel Module Programming Guide*. Accessed: 2025-12-15. 2007. URL: `https://tldp.org/LDP/lkmpg/2.6/html/index.html`.

AI in IT and Business Project Management: Insights Across the Project Management Life Cycle*

Panel Discussion

Aziz Fellah, Sheng Chai, Cindy Tu, Yu Zhao
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, MO 64468
{afellah, schai, cindytu, zhao}@nwmissouri.edu

1 Summary

This study investigates how Generative AI, including ChatGPT, Gemini, Groq, and Claude, can enhance the full Project Management Life Cycle (PMLC) in two courses: the undergraduate course, *IT Project Management*, and the graduate course, *Project Management in Business and Technology*. The work is grounded in a classroom-based setting where Generative AI is used by students not as a replacement for student decision-making, but as an assistive and comparative tool across all phases of the ten Project Management Knowledge Areas: Integration, Scope, Schedule, Cost, Quality, Resource, Communications, Risk, Procurement, and Stakeholder Management.

Student teams assumed dual roles as stakeholders and developers, creating diverse projects by defining their own projects as a team, such as websites, mobile apps, and games spanning multiple domains, including education, healthcare, and entertainment. All projects were then randomly assigned to other teams, with each team selecting its own project manager to oversee planning, execution, and control. Each project was evaluated across all ten Project Management Knowledge Areas, and comparisons were made between student-driven and AI-generated approaches.

Findings indicate that AI excels in structured tasks, such as scheduling, cost estimation, and risk analysis, but is less effective in areas requiring creativity,

including game development, complex stakeholder management, and adaptive decision-making. These results highlight the complementary strengths of AI and student approaches, demonstrating the practical impact of AI-supported project management in educational environments that reflect real-world constraints, stakeholder ambiguity, and diverse project domains.

2 Student-Driven vs AI-Generated Project Management

First, students worked collaboratively in teams across the ten Project Management Knowledge Areas, using their own problem-solving skills, domain knowledge, and communication practices, without AI involvement. Relying on the techniques, strategies, and lectures presented in class, teams employed project management and control tools such as GitHub and Jira to support collaboration, tracking, and project oversight. This student-driven phase captured how students performed across the PMLC using classical project management techniques, based on the five process groups and ten knowledge areas outlined in the Project Management Body of Knowledge (PMBOK).

Next, students applied Generative AI tools to support each phase of the PMLC and each Project Management Knowledge Area, freely selecting tools such as ChatGPT, Gemini, Groq, and Claude. AI was used for tasks including requirements elicitation, planning, scheduling, risk identification, communication planning, and project documentation, with AI-generated outputs considered support artifacts. This phase enabled a direct comparison between student-driven and AI-generated outputs across all knowledge areas.

Students then compared student-generated artifacts with AI-generated outputs across all ten knowledge areas, examining accuracy, completeness, contextual relevance, creativity, and adaptability. The results indicate that AI tools perform differently depending on the knowledge area; for example, Claude excelled in communication tasks, Gemini in scheduling and planning, and ChatGPT in requirements-related activities. While Generative AI performed well in structured, documentation-heavy tasks, it was less effective in areas requiring creativity and adaptive decision-making, particularly in game development projects.

Findings highlight the complementary strengths of AI and student approaches, demonstrating the impact of AI-supported project management in educational settings with real-world constraints, stakeholder ambiguity, and diverse domains. This work invites discussion on integrating Generative AI across the PMLC, understanding the strengths of students and AI, and balancing AI-assisted efficiency with student-centered project management.

3 Biographies

Aziz Fellah is an Associate Professor of Computer Science with extensive experience across multiple institutions. His background spans several subfields within the discipline of computer science.

Sheng Chai is an Assistant Professor of Computer Science actively involved in research on high-dimensional data analysis and information retrieval.

Cindy Tu is an Associate Professor of Information Systems and Graduate Coordinator, with a focus on information systems and business intelligence.

Yu Zhao is an Assistant Professor of Computer Science with expertise in data analytics and information security.

Rethinking the AI Syllabus: Designing Traffic-Light Policies in CS Courses That Do Not Backfire*

Workshop

Usamah Moin Mohammed
Computer Science Department
Lincoln University of Missouri
Jefferson City, MO 65101
mohammedu@lincolnu.edu

Workshop Category

Workshop (hands-on, 90–120 minutes)

Abstract

Generative AI tools such as ChatGPT, Gemini, Copilot, and Claude have changed how students write, code, and study. While many universities now provide high-level policy guidance, individual instructors often struggle to translate these broad rules into clear, assignment-level instructions. The result can be confusion, inconsistent enforcement, and policies that inadvertently punish legitimate tool use.

This workshop presents a practical, classroom-tested framework for AI policies in CS courses built around two core ideas: (1) AI use is restricted by default and only permitted when clearly allowed, and (2) every assignment is labeled with a traffic-light status: Red (closed to AI), Yellow (restricted AI use with required disclosure), or Green (AI-integrated work). We move beyond unreliable detection tools to focus on process, distinguishing between "correction" tools (e.g., spell-check) and "creation" tools. Participants will leave with ready-to-use policy paragraphs and a traffic-light scheme tailored to their own specific assignments.

Extended Description

1. Goals and Rationale

This workshop is for CS faculty who need to move beyond generic institutional statements to create defensible, operational AI policies. While some institutions provide default language, this workshop addresses the gap between high-level policy and assignment-level reality.

Goals:

- Give instructors a concrete, defensible AI policy structure they can actually enforce.
- Help them distinguish between everyday correction/accessibility tools and generative tools that do real intellectual work.
- Provide a simple assignment-level scheme (Red / Yellow / Green) that students understand immediately.
- Address the limitations of AI detection tools by shifting focus to "cognitive responsibility" and process documentation.

2. Intended Audience and Format

Audience: Teaching-focused CS faculty, lecturers, and instructors; program coordinators; interested K–12 CS teachers. No AI research background required.

Format: Hands-on workshop, 90–120 minutes.

Assumptions: Participants bring at least one existing assignment, project, or exam (printed or digital).

3. Workshop Outline

10–15 min **Framing:** Overview of the traffic-light model and the restricted by default principle. Discussion on why reliance on detection software is often insufficient.

15–20 min **Components:** Walk through a compact AI policy template: Default Rules, Correction vs Creation, Assignment-Level Rules (Red/Yellow/Green), Cognitive Responsibility, and Accessibility.

10–15 min **Examples:** Show how the policy appears in different courses (intro vs upper-level). Show how one assignment is labeled Red, another Yellow, and a project Green.

40–55 min **Hands-on Redesign:**

1. Step 1: Classify each chosen assignment as Red, Yellow, or Green.
2. Step 2: Write a short AI-use box to attach to that assignment.
3. Step 3: Add a simple AI-use reflection prompt where appropriate.

10–15 min **Wrap-up:** Share-out of redesigned assignments. Discussion of edge cases (large classes, online exams) and troubleshoot participant-specific scenarios.

4. Takeaways

Participants leave with:

- A copy of a full AI policy template discussed in the workshop.
- Short, copyable syllabus paragraphs for default AI clauses and assignment-level traffic-light labels.
- At least one redesigned assignment or exam handout with explicit AI rules.
- A brief list of strategies for AI enforcement that avoid over-relying on detectors.

5. Equipment and Logistics

- Standard projector and screen.
- Whiteboard or flip chart for group examples.
- Participants are encouraged to bring a laptop or tablet for editing their own documents.