The Journal of Computing Sciences in Colleges

Papers of the 26th Annual CCSC Northwestern Conference

October 11th and October 12th, 2024
Willamette University
Salem, OR

Abbas Attarwala, Editor
California State University, Chico

Sharon Tuttle, Regional Editor
Cal Poly Humboldt

Volume 40, Number 1

October 2024

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

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.

Gold Level Partner
Rephactor
ACM2Y
Code Grade
GitHub

Welcome to the 2024 CCSC Northwestern Conference Willamette University

On behalf of the Conference Steering Committee and the Northwest Regional Board, it is my pleasure to extend a warm welcome to all attendees of the Twenty-Sixth Annual Consortium for Computing Sciences in Colleges (CCSC) Northwestern Regional Conference. We are honored to host this year's event at Willamette University and are eagerly looking forward to engaging with you all in person.

I would like to extend my sincere gratitude to the many individuals and groups who contributed their time and effort to coordinating and supporting this year's conference. Their dedication has been invaluable. This year's program features 11 peer-reviewed papers, six panels and tutorials, and a keynote address. All submissions underwent a rigorous peer review process, with 12 out of 18 papers being accepted, resulting in an acceptance rate of 67%.

We would like to express our sincere appreciation to our colleagues from across the region who served as professional reviewers. Their generous contributions of time and expertise were invaluable in shaping the selection of this year's conference program. Our Keynote speaker, Selena Deckelmann, Chief Product and Technology Officer from Wikimedia Foundation, will discuss how Wikipedia utilizes AI.

Of course, none of this would be possible without the support of our national and local partners. A heartfelt thank you also goes out to you, the attendees, whose participation is crucial not only to the success of conferences like this, but also to fostering continued communication and collegiality among those dedicated to advancing and promoting our discipline.

We are thrilled to welcome you to our campus and hope you enjoy the conference, as well as the opportunity to connect with colleagues at this annual gathering.

Lucas Cordova
Willamette University
CCSC-NW 2024 Conference Chair

Humans in the Loop: How Wikipedia's AI Tools Keep Volunteers at the Helm[*]

Keynote

Selena Deckelmann
Chief Product and Technology Officer at the Wikimedia Foundation

For over two decades, volunteers around the world have contributed their time and expertise to a revolutionary project: Wikipedia, the online encyclopedia anyone can edit. Wikipedia has grown with the internet and, thanks to these volunteers, is now recognized as the largest digital repository of knowledge ever.

With the recent cascade of accessible generative AI tools, the internet has entered a new era — one full of possibilities, but also uncertainty about how broad implementation of these tools will impact the human-built web we know and depend on for accurate, reliable information.

Selena Deckelmann, Chief Product and Technology Officer at the Wikimedia Foundation, will speak to how Wikipedians use AI advancements to support, not replace, human contributions across the Wikimedia projects, and what lessons can be gleaned from Wikipedia's open model of global collaboration and connection.

[*]Copyright is held by the author/owner.

Computer Science Assignments and Activities that Support Academic Belonging[*]

Panel Discussion

Shereen Khoja[1], Tammy VanDeGrift[2]
[1]Mathematics, Computer Science, and Data Science
Pacific University, Forest Grove, OR 97116
shereen@pacificu.edu
[2]Computer Science, Shiley School of Engineering
University of Portland, Portland, OR 97203
vandegri@up.edu

1 Abstract

Feeling a sense of belonging is an essential human need that, when met, is associated with positive outcomes [1]. Academic belonging is defined by Lewis et al as the extent to which students subjectively perceive that they are valued, accepted, and legitimate members in their academic domain [2], and a sense of academic belonging in a course has been consistently linked to engagement in STEM coursework [3].

Faculty members will share assignments and activities that they use to foster a sense of belonging in their computer science courses. Each panelist will provide an overview of the activity or assignment, its goal(s) or learning objective(s), and any outcomes and observations. After the panelists present, attendees will be invited to share their ideas and activities they use to foster a sense of belonging in a computer science classroom.

The assignments and activities fall under the following course and classroom practices that have been shown to foster a sense of belonging in higher education:

- **Fostering supportive spaces:**

[*]Copyright is held by the author/owner.

  - Blobs/lines activity.
  - Thanking those who support others in group work.
  - Allowing multiple modalities for collaborative work in the classroom (computer, whiteboard, paper).
- **Sharing authentic self:**
  - Sharing personal journey in CS.
  - Giving students options for how to present projects (infographic, video, paper, etc.).
  - Start of semester survey where I share about myself and invite students to share what they feel comfortable sharing about themselves.
- **Embracing an asset-based (anti-deficit) teaching approach:**
  - Artwork to demonstrate a computing concept.
  - In software engineering, students articulate their assets and we discuss and apply these to their senior capstone projects.
- **Cultivating growth mindset:**
  - Exam reflections.
  - Share stories of my own struggles when I was a student.
  - Low stake assignments in upper division courses that encourage seeking critique.
- **Avoiding reinforcing stereotypes of who belongs or does not belong in a discipline:**
  - Students share non-computing interests.
  - Start of semester survey where I share part of my personality or hobby that does not fit the stereotype.

References

[1] Roy F. Baumeister and Mark R. Leary. The need to belong: Desire for interpersonal attachments as a fundamental human motivation. *Psychological Bulletin*, 117(3):497–529, 1995.

[2] Karyn L. Lewis, Jane G. Stout, Steven J. Pollock, Noah D. Finkelstein, and Tiffany A. Ito. Fitting in or opting out: A review of key social-psychological factors influencing a sense of belonging for women in physics. *Physical Review Physics Education Research*, 12(2):020110, 2016.

[3] Denise Wilson, Diane Jones, Fraser Bocell, Joy Crawford, Mee Joo Kim, Nanette Veilleux, Tamara Floyd-Smith, Rebecca Bates, and Melani Plett. Belonging and academic engagement among undergraduate stem students: A multi-institutional study. *Research in Higher Education*, 56:750–776, 2015.

An Introduction to MPICH Parallel Programming in C++[*]

Conference Tutorial

Xuguang Chen
Computer Science Department
Saint Martin's University
Lacey, WA 98503
xchen@stmartin.eduu

Parallel Computing and MPI

Parallel computing is a type of computation where many calculations or processes are carried out simultaneously. Hence, a large problem can often be divided into smaller subtasks that then can be solved at the same time so as to increase computational speed and efficiency, leveraging multiple processors or computers working together. Parallel computing has been applied to a variety of application area such as scientific computing, engineering simulations, image processing, and big data processing.

Parallel programming models are diverse, with two of the most well-known being the message-passing model and the shared-memory model. As an example of the message-passing model, the Message Passing Interface (MPI) is a communication protocol and library standard proposed for parallel computing. It allows different nodes in a distributed computing environment to communicate and coordinate with each other by explicitly sending and receiving messages. MPI includes various operations, for example:

1. Initialization and Termination: `MPI_Init` and `MPI_Finalize`
2. Point-to-Point Communication: `MPI_Send` and `MPI_Recv`
3. Collective Communication: broadcast (`MPI_Bcast`), reduction (`MPI_Reduce`), and scatter (`MPI_Scatter`)

[*]Copyright is held by the author/owner.

4. Synchronization: `MPI_Barrier`

As a protocol and library standard, MPI can be implemented in different languages, including C, C++, Java, Python, and Fortran.

MPICH

MPICH is a high performance and widely portable implementation of the MPI standard. It supports both point-to-point and collective communication protocols, making it suitable for a wide range of applications, for instance development of parallel applications running simultaneously on multiple processors or computers to effectively handling large-scale computations, academic and industrial research to explore new parallel algorithms and improve existing ones, testing and benchmarking the performance of parallel systems and applications so as to provide insights into scalability and efficiency, and deployed in various high-performance computing (HPC) systems and supercomputers to perform complex simulations, data analysis, and scientific computations.

Tutorial Overview

MPICH can be implemented in various programming languages. This tutorial focuses on using MPICH with C++. It will begin by explaining where to obtain and how to install the necessary software on both Windows and Linux machines. Next, the tutorial will guide the audience through editing, compiling, and running a C++ MPICH program on these operating systems. Following this, the basic structure of an MPI program will be outlined. The tutorial will then cover essential MPI routines for point-to-point and collective communications, along with their implementations in MPICH. Finally, it will provide a list and description of materials suitable for further study.

This tutorial is designed for beginners in parallel programming and those who are interested in MPI programming. Additionally, many universities and colleges have offered parallel computing classes or incorporated parallel computing topics into other related courses. Hence, this tutorial can also be served as a module within such course, introducing the basic concepts and operations of parallel computing to students. It assumes that participants have a basic understanding of programming languages, especially Java, C#, or C++.

As the learning outcomes, participants should be able to achieve the following:

1. Understand the necessary software for MPI parallel programming with MPICH, including how to obtain and install it on Windows and Linux machines.

2. Gain knowledge about MPI, especially the concepts of point-to-point and collective communications.
3. Learn how to edit, compile, and run an MPI program in C++ to implement basic MPI operations.

At the end of the tutorial, an e-version of the lecture notes, example C++ code, and other materials for self-study can be provided upon request.

References

[1] MPI Tutorial Website. `https://mpitutorial.com/tutorials`.

[2] MPICH Official Website. `https://www.mpich.org`.

[3] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing.* Pearson Education. Addison-Wesley, 2003.

[4] W. Gropp, T. Hoefler, R. Thakur, and E. Lusk. *Using Advanced MPI: Modern Features of the Message-Passing Interface.* Scientific and Engineering Computation. MIT Press, 2014.

[5] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-passing Interface.* Number v. 1 in Scientific and engineering computation. MIT Press, 1999.

[6] P. Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann, 1997.

[7] T. Rauber and G. Rünger. *Parallel Programming: for Multicore and Cluster Systems.* Springer Berlin Heidelberg, 2013.

[8] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The Complete Reference.* Scientific and engineering computation series. MIT Press, 1998.

Debugging Server- and Client-Side Javascript[*]

Conference Tutorial

Tiva Rocco, Joshua B. Gross
California State University, Monterey Bay
Seaside, CA 93955
{trocco,jgross}@csumb.edu

Abstract

Tutorial and hands-on demonstration of debugging tools and techniques for Javascript and Node.js applications, providing common errors and practical solutions to improve code quality and development efficiency.

Introduction

Node.js is a runtime environment that executes Javascript code outside of a browser, often used for building server-side web applications. Many Node.js and Javascript tutorials provide correct code implementation and compilation, but few address techniques and tools for finding and correcting errors. Debugging server-side code can be especially difficult because it may not always be clear if errors are occurring on the host or client side.

We will demonstrate setting up debugging environments and identifying and handling common errors. Participants will learn to use server-side and client-side debugging tools by engaging in live debugging exercises to reinforce their understanding. We will have a public Github repository prepared with a tutorial guide, sample programs, and solutions. Examples of debugging scenarios are derived from existing research[1] and previous experience.

[*]Copyright is held by the author/owner.

Presenter Biographies

Tiva Rocco is an undergraduate student at CSU-Monterey Bay, pursuing a BS in Computer Science with a concentration in Software Engineering. She has served as a Teaching Assistant for the Internet Programming course for multiple semesters, and is a coordinator for the computer science department TA program, mentoring and supporting computer science TAs, conducting training sessions, and contributing to the continuous development of the program. Tiva has also dedicated over 50 hours of community service related to software engineering for organizations like the UC-Santa Cruz Life Lab and Community Builders for Monterey County.

Joshua B. Gross, PhD is an associate professor of computer science at California State University Monterey Bay. He teaches courses on introductory programming, data structures, and databases. His research is in the area of computer science education, specifically focused on helping students better understand computational complexity.

Approach

The tutorial will consist of:

- Installing software (all software is free and available for Mac, Windows, and Linux)
- Example code with common types of errors
- Guided instruction on how to use different tools and techniques to identify and resolve errors
- Activities for participants to practice their debugging skills

Intended Audience

This tutorial is targeted at computer science students, developers, and faculty who work with Javascript and Node.js applications, and who would benefit from learning effective debugging techniques and solutions to common errors. Participants should be familiar with basic concepts of server-side web development. Familiarity with JavaScript language basics would be beneficial but not required, because the course material will include code for all examples.

Computer Requirements

Computers running Windows, macOS, or Linux with an internet connection will be able to participate in the hands-on demonstration. Participants should be comfortable using a browser, text editor, git, and a command line for their system.

Participants will install Microsoft Visual Studio Code (available free for all platforms), the Firefox browser, and various JavaScript libraries, including Node.js.

References

[1] Quinn Hanam, Fernando S de M Brito, and Ali Mesbah. Discovering bug patterns in javascript. In *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, pages 144–156, 2016.

Bridging the Quantum Computing Skills Gap: Integrating Quantum Education into Computer Science Curricula*

Conference Tutorial

Farzin Bahadori and Radana Dvorak
Department Of Computer Science
Hal and Inge School of Engineering
Saint Martin's University
Lacey, WA 98503
{fbahadori,rdvorak}@stmartin.edu

The lack of a skilled workforce in quantum computing parallels the current cybersecurity skills gap. [7], [6], [8] To prevent a similar gap in quantum computing as currently industry is experiencing in filling cybersecurity jobs, we present proactive education strategies to address the issue. This tutorial will present current efforts in quantum education and how Saint Martin's Computer Science Department contributes to these efforts by integrating quantum computing modules into existing curricula using the KEEN Entrepreneurial Mindset framework [2]. Participants will be able to implement one of the introductory models utilizing open-source tools and resources to engage students in solving problems using quantum computing concepts.

The tutorial will address: (1) the parallels between the cybersecurity and quantum computing skills gaps. (2) explore the role of early and comprehensive curricular integration in mitigating skills deficits, (3) present the KEEN Entrepreneurial Mindset framework as a tool for enhancing quantum computing education to instill curiosity about our evolving world of work encouraging students to adopt a critical approach exploring alternative solutions to today's problems, and (4) share practical strategies for incorporating introductory quantum computing modules into computer science curricula.

By the end of this tutorial, participants will be able to analyze the effectiveness of different educational initiatives aimed at bridging these skills gaps in

*Copyright is held by the author/owner.

quantum computing; apply the KEEN Entrepreneurial Mindset (EM) [2], and design and implement one of the introductory quantum computing modules [5].

The tutorial content will include the following:

- Educational Framework introduces the KEEN Entrepreneurial Mindset framework and how the three C's (Curiosity, Connections, Creating Value) foster student engagement.
- Curriculum Development explaining how the quantum computing modules were designed for computer science students without strong mathematics and physics background. [4], [5]
- Implementation strategies outlines how quantum computing can in incorporated in existing courses utilizing open-source tools and resources to provide hands-on experience such as a brief introduction to Quantum and Gates; setting Quantum Simulators on a personal computer; overview of the open source tools and the IBM lab and composer; how to work with qubits and creating superposition. [1], [3], [5]

References

[1] IBM Commits to Skill 30 Million People Globally by 2030. `https://newsroom.ibm.com/2021-10-13-IBM-Commits-to-Skill-30-Million-People-Globally-by-2030`.

[2] KEEN - The Framework. `https://engineeringunleashed.com/mindset-matters/framework.aspx`.

[3] QUIST. `https://qis-learners.research.illinois.edu/`.

[4] Sophia Chen. The future is quantum: universities look to train engineers for an emerging industry. *Nature*, 623(7987):653–655, 2023.

[5] Radana Dvorak and Farzin Bahadori. Leveraging open source tools to teach quantum computing foundations: Bridging the future workforce gap in the quantum era. In *2024 ASEE Annual Conference & Exposition*, 2024.

[6] Michael FJ Fox, Benjamin M Zwickl, and HJ Lewandowski. Preparing for the quantum revolution: What is the role of higher education? *Physical Review Physics Education Research*, 16(2):020131, 2020.

[7] Owen Hughes. Fixing the next big tech skills shortage will need a quantum leap. `https://www.zdnet.com/article/fixing-the-next-big-tech-skills-shortage-will-need-a-quantum-leap`.

[8] Stefan Seegerer, Tilman Michaeli, and Ralf Romeike. Quantum computing as a topic in computer science education. In *Proceedings of the 16th Workshop in Primary and Secondary Computing Education*, pages 1–6, 2021.

Designing Contributing Guides to Facilitate Undergraduate Software Development Research[*]

Conference Tutorial

Anna Ritz, Oliver Anderson, Altaf Barelvi
Biology Department
Reed College
Portland, OR 97202
{aritz,aoliver,abarelvi}@reed.edu

With the rise of collaborative scientific research, new researchers must quickly and easily contribute to projects. Contributing guides are specialized tutorials in the software development industry that help new developers with minimal coding experience quickly contribute to a project's goals. Contributing guides differ from general tutorials in their purpose: rather than focusing on getting a user started with a software tool, contributing guides focus on facilitating contributions to the development of a tool. During undergraduate summer research experiences in computer science, students join long-term ongoing projects and contribute code to them, often within a limited time frame of ten weeks or less. Thus, contributing guides are uniquely suited for undergraduate summer research projects, as they facilitate quick new contributions to existing software tools.

In this tutorial, we show best practices for writing contributing guides based on two examples of contributing guides from computational network biology projects in an undergraduate summer research experience setting: ProteinWeaver and Signaling Pathway Reconstruction Analysis Streamliner (SPRAS). Both projects contain the key elements of a successful contributing guide. First, new collaborators must familiarize themselves with the project's overarching goal. Then, the guide should provide collaborators with the site's basic architecture and tutorials for the software tools used to develop the project. Once new contributors are familiar with the working parts of the project, the goal of the contributing guide is to get new developers to add to the codebase by

[*]Copyright is held by the author/owner.

slightly modifying the most relevant parts of the code to the project. The tutorial highlights key components of a contributing guide for summer undergraduate research experiences in software development.

Acknowledgements

We thank Dr. Tony Gitter (UW-Madison) for being the first to design the SPRAS Contributing Guide. We also thank Gabe Howland and Schwa Yeleti for helping improve the contributing guides as summer researchers. This work was funded by NSF #1750981.

Cybersecurity Exercises in the Age of LLMs*

Conference Tutorial

Richard Weiss[1], Jens Mache[2]
[1]The Evergreen State College
Olympia, WA 98505
weissr@evergreen.edu
[2]Lewis & Clark College
Portland, OR 97219
jmache@lclark.edu

In this tutorial, we will introduce a cybersecurity education framework for developing polymorphic hands-on exercises. Many faculty readily acknowledge the importance of cybersecurity in the Computer Science curriculum, but there are still barriers to integrating it into existing courses. One of those barriers is the fact that in most courses, the current content fills the entire term. Another issues is that faculty don't have time and expertise to create new content that would fit well with their current content and style. The third problem is that exercises created should be resistant to solution by LLMs. We have developed cybersecurity exercises that combine two principles: environment specificity and polymorphism. Environment specificity means that the solutions to the exercise should depend on the local environment (LLMs don't have access to that information). In this context, polymorphism means that they can be easily modified each time that the class is taught.

Overview

EDURange [2, 6, 3, 7, 5] has been developed over more than ten years, and it continues to evolve. We have used it to integrate hands-on security exercises in our own classrooms and will present our exercises and framework that satisfy the two principles: environment specificity and polymorphism. The exercises that we will describe are relevant to: introductory courses that use the Linux command line, Operating Systems, Computer Networking, Database Systems.

*Copyright is held by the author/owner.

We will also demonstrate the feedback system that we have added that allows instructors to interact with students one on one [9, 4, 8, 1].

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 2216485 and 2216492.

References

[1] Aubrey Birdwell, Jack Cook, Richard Weiss, and Jens Mache. From logs to learning: Applying machine learning to instructor intervention in cybersecurity exercises. In *Proceedings of the American Society for Engineering Education (ASEE) Annual Conference*, 2024.

[2] Jack Cook, Richard Weiss, Jens Mache, Carlos García Morán, and Justin Wang. An authoring process to construct docker containers to help instructors develop cybersecurity exercises. *Journal of Computing Sciences in Colleges*, 37(10):37–47, 2022.

[3] Jelena Mirkovic, Aashray Aggarwal, David Weinman, Paul Lepe, Jens Mache, and Richard Weiss. Using terminal histories to monitor student progress on hands-on exercises. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 866–872, 2020.

[4] Quinn Vinlove, Jens Mache, and Richard Weiss. Predicting student success in cybersecurity exercises with a support vector classifier. *Journal of Computing Sciences in Colleges*, 36(1), 2020.

[5] Richard Weiss, Stefan Boesen, James F. Sullivan, Michael E. Locasto, Jens Mache, and Erik Nilsen. Teaching cybersecurity analysis skills in the cloud. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, page 332–337, New York, NY, USA, 2015. Association for Computing Machinery.

[6] Richard Weiss, Franklin Turbak, Jens Mache, and Michael Locasto. Cybersecurity education and assessment in edurange. *IEEE Security & Privacy*, May/June, 2017.

[7] Richard Weiss, Franklin Turbak, Jens Mache, Erik Nilsen, and Michael E Locasto. Finding the balance between guidance and independence in cybersecurity exercises. In *2016 USENIX Workshop on Advances in Security Education (ASE 16)*, 2016.

[8] Valdemar Švábenský, Kristián Tkáčik, Aubrey Birdwell, Richard Weiss, Ryan S. Baker, Pavel Čeleda, Jan Vykopal, Jens Mache, and Ankur Chattopadhyay. Detecting unsuccessful students in cybersecurity exercises in two different learning environments. In *Proceedings of the IEEE Frontiers in Education Conference (FIE)*, 2024.

[9] Valdemar Švábenský, Richard Weiss, Jack Cook, Jan Vykopal, Pavel Čeleda, Jens Mache, Radoslav Chudovský, and Ankur Chattopadhyay. Evaluating two approaches to assessing student progress in cybersecurity exercises. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, 2022.

Investigating Hinge-Points in Computer Science Identity and Career Plans*

Ben Tribelhorn[1] and Nicole Ralston[2]
[1]Shiley School of Engineering
[2]School of Education
University of Portland
Portland, OR 97203
{tribelhb, ralston}@up.edu

Abstract

Computer Science (CS) identity is a topic of rising concern as many believe it is a key contributor to student success. We implemented a meta-survey to search for events and perceptions that might impact CS identity and students' career plans relating to CS.

We found significant differences in CS identity based on the course level. Across all respondents we found significant improvement in a performance/competence sub-construct and a recognition sub-construct that suggest applied computing skills are the area students see their improvement across all courses. This is furthered by an analysis of qualitative responses that suggests CS faculty have an opportunity to shape the formation of CS identity.

1 Introduction

Student identity is becoming a more frequently studied topic, and is relevant to Computer Science due to common issues at most institutions. Computer Science (CS) programs are heavily gender skewed and are often still pervaded by specific stereotypes that reduce inclusivity.With typical concerns of retention and adapting to the ever evolving discipline, especially considering the

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

impacts of stereotype threat[1], it is timely to consider curriculum design and implementation's impact on forging a CS specific identity.

The authors' motivation for this study was to embark on a longitudinal study of identity and probable causes that impact the development of a CS identity, both positively and negatively, across the CS disciplinary courses at our institution. Identifying hinge points and seminal experiences will help the faculty improve courses with a more holistic focus. This study used two validated instruments to collect data from our CS program. We measure identity with Mahadeo et al.'s tool aimed at helping researchers and practitioners improve student persistence in computer science[6]. Further, we use components of Hoegh & Moskal's survey to look at additional variables that could help improve recruitment and retention[3].

2 Background

Rodriguez & Lehman advocate for a sociocultural perspective on analyzing identity development in order to create more equitable computing outcomes[7]. Further they suggest that creating greater theoretical understanding of the computing identity development process will help inform CS educational practices.

A small study by Kapoor & Gardner-McCune investigated the development of student's professional identity and found that CS undergraduates tend to form this between years 2-3 of their degree programs[4]. Beyond intrinsic and discipline-specific factors, they found that professional identity is also shaped by activities such as coursework, informal activities like hackathons, and professional development activities including internships and conferences.

Lunn et al. studied the impact of experiences such as technical interviews' effect on identity[5]. They observed that for the students that did not have positive experiences with technical interviews they had a reduced computing identity, but that facing discrimination during technical interviews had the opposite effect. They also noted that having friends in computing bolsters computing identity for Hispanic/Latino-a students, as does a supportive home environment for women. We interpret these results to suggest that one curricular intervention for positive identity formation would be to focus on technical interview preparation.

Finally, very recent work by Fong et al. has looked at how collaborative learning courses impact a sense of belonging[2]. They find that collaborative work does increase a sense of belonging and is malleable beyond the first year experience. In their study they intervened to assign a majority of women to groups in most of the courses studied, (allowing for groups of only men), which was not disentangled from the collaboration intervention.

The literature shows that studies of identity have found some activities that impact a computing identity, but there is not yet a clear line of research that illustrates where these identities are shaped within the curriculum and other areas of faculty intervention, such as office hours, club advising, etc. This paper seeks to report initial results focused on locating these impactful experiences.

3 Research Questions and Methods

These goals led to the following research questions that might be answered using previously validated survey items.

RQ1: To what extent does CS identity change across the curriculum?

RQ2: What events and perceptions relate to CS identity?

RQ3: To what extent do students plan to pursue CS careers and how does that change?

3.1 Survey implementation

We generated a survey, using a combination of questions from two prior studies and additional open ended text responses, as both a pre and post instrument at the beginning and end of our CS 1 (3 sections), CS 2 (3 sections), and CS Seminar courses (2 sections), in addition to the beginning of the Fall Capstone and end of Spring Capstone during the AY 2023-24.

The Computer Science program teaches Java in our CS 1 and does a semester long project in Android in our CS 2 (using Java). Students may take the Data Structures in C course before, concurrent, or after CS 2. Typically they take CS 2 ahead of Data Structures, so surveying these two courses captures early CS majors and minors in addition to the "CS-interested" on a space available basis. The ethics and professionalism focused CS Seminar course requires junior standing, and does not count towards the minor. Finally, we survey Capstone students, but due to an unforeseen programmatic change this led to a low completion rate for these students making it difficult to see significant results from the Capstone perspectives.

3.2 Participants

In sum, 104 students completed the survey at pre and 65 students completed the survey at post; thus results should be interpreted cautiously as all students did not complete at both time points and because the surveys were conducted anonymously to increase honesty and reduce bias the pairs could not be matched. Students completed the survey as part of four different classes, CS 1 (n = 56, 33%), CS 2 (n = 29, 17%), CS Seminar (n = 55, 33%), and Capstone (n = 29, 17%). Only those identifying as computer science majors or computer science minors were included.

4 Results

4.1 Computer Science Identities

First, students were asked the extent to which they identified as computer scientists on a scale of 1 (do not at all identify) to 10 (fully identify) on a sliding scale. These were completed at two time points in each class – at the beginning of the class (pre) and at the end of the class (post). A two-way ANOVA revealed there were overall statistically significant differences regardless of the time period by course, F(3, 161) = 5.88, $p < .001$, with those in CS 1 (the introductory level course) scoring significantly lower than all other courses. The two-way ANOVA also revealed a statistically significant interaction effect, F(3, 161) = 3.29, p = .022. This exhibited as the students with the least amount of experience with computer science in the lowest level course (CS 1) experienced the largest amount of change from pre to post, growing about 1.11 points, however the other three courses experienced overall drops from pre to post (see Table 1) which could have been from a lower completion rate at post biasing responses.

Table 1: To What Extent Do You Identify as a Computer Scientist?

	Pre		Post	
Course	*n*	*M*(*SD*)	*n*	*M*(*SD*)
CS 1	36	5.14 (2.94)	20	6.80 (2.40)
CS 2	19	7.63 (1.92)	10	7.10 (2.51)
CS Seminar	31	7.71 (1.42)	24	7.33 (2.24)
Capstone	18	8.22 (1.48)	11	7.09 (1.56)
Total	104	6.89 (2.49)	65	7.09 (2.20)

Next, to further understand computer identity, the responses to the nine Computer Identity survey questions were summed to understand how much computer identity these students felt[6]. Because individual items were on a 4-point Likert scale ranging from Strongly Disagree (1) to Strongly Agree (4), the total possible was 36 (9 times 4). Table 2 below shows the means and standard deviations for these total scores again by time period and course. On this instrument, the scores overall improved from pre to post for the two lower-level courses, however remained more stagnant for the two upper-level classes, which a two-way ANOVA revealed to be statistically significant differences by class F(3, 161) = 3.43, p = .018. Overall, students felt fairly high computer identity, with average post scores of 26.79, about 74% of the possible total of 36, or an average score of about 'Agree' on the Likert scale for each of the 9 items.

Table 2: Total Sum Scores on Computer Identity Instrument

	Pre		Post	
Course	*n*	*M*(*SD*)	*n*	*M*(*SD*)
CS 1	36	22.22 (7.14)	20	25.50 (7.76)
CS 2	19	26.47 (5.45)	10	28.50 (5.95)
CS Seminar	31	27.13 (4.57)	24	27.33 (5.10)
Capstone	18	26.50 (5.09)	11	26.36 (7.02)
Total	104	25.20 (6.14)	65	26.79 (6.40)

Next, we sought to dive deeper into the individual items on the instrument at pre and post across all of the different classes. A few interesting findings arise in Table 3 below. First, two of the items demonstrated statistically significant changes from pre to post via independent samples t-tests. The first was *I can do well on computing tasks (e.g., programming and setting up servers)*, t(167) = 3.120, p = .002, which grew almost half a point from pre to post. The second item was *I understand concepts underlying computer processes*, t(167) = 2.614, p = .01, which grew about a third of a point from pre to post. Last, the item, *My instructors/teachers see me as a computer-savvy person* experienced marginal significance, t(167) = 1.967, p = .051, also growing about a third of a point from pre to post. Notably, both of the significant items were part of the Performance/Competence sub-construct.

Most of the remaining items also experienced growth from pre to post, except two items that experienced declines, which were all related to interest in computer science: *I like to peruse forums* and *Computer programming is interesting to me.* It seems that the more students learn, the less interested they become in the topic?! Or perhaps being tired at the end of the semester has reduced their interest.

Lastly of notice, most of the results are high (close to 3 on a 4-point scale) however these interest items (*Computer programming is interesting to me* and *Topics in computing excite my curiosity*) were also the highest at pre and post. Besides the perusing forums item already discussed, the lowest items at post were *I understand concepts underlying computer processes, My instructors/teachers see me as a computer-savvy person* (even after experiencing a third of a point of growth), and interestingly, *Others ask me for help with software (applications/programs).* These take-aways may have implications for programming, in both continuing to incite curiosity in our courses and ensuring our students understand these complex processes and feel confidence and self-efficacy in these worlds.

Table 3: Individual Survey Item Mean Scores: Computer Identity Instrument

Course	Pre n=104 *M*(*SD*)	Post n=65 *M*(*SD*)
(r) My family sees me as a computer-savvy person.	3.18 (1.09)	3.45 (0.90)
(r) Topics in computing excite my curiosity.	3.28 (0.77)	3.35 (0.76)
(pc) I can do well on computing tasks (e.g., programming and setting up servers).	2.59 (0.87)	3.02* (0.87)
(r) My friends/classmates see me as a computer-savvy person.	2.76 (0.97)	2.89 (0.95)
(i) I like to peruse forums, social media, or online videos about computer-related topics.	2.62 (1.04)	2.57 (0.98)
(pc) I understand concepts underlying computer processes.	2.50 (0.89)	2.86* (0.85)
(r) My instructors/teachers see me as a computer-savvy person.	2.25 (1.08)	2.59** (1.07)
(i) Computer programming is interesting.	3.44 (0.72)	3.42 (0.68)
(pc) Others ask me for help with software (applications/programs).	2.62 (1.20)	2.65 (1.35)

Notes: r = Recognition sub-construct. i = Interest sub-construct. pc = Performance/Competence sub-construct. *$p < .05$, **$p = .051$.

Students were asked to elaborate through open-ended responses about concepts:

- "I feel like I do not have a thorough understanding of a few fundamental topics, but rather a rushed, minimal understanding of many different things; many of which are self-learned as they are only brushed on in classes."
- "I am well known to the people around me to be very good with computers. Certainly, I act as the local IT person for my parents, grandparents, and less tech-savvy siblings. I constantly try to cultivate and expand my knowledge of computing, and I spend a lot of time reading and thinking about it."
- "As someone who came to [University of Portland] not liking Computer Science my thought on the subject has completely changed after taking my first computer science class at University of Portland. In fact, I enjoy just thinking of ways to code something in my head without typing it on the regular."

4.2 Computer Science Career Plans

After understanding computer science identities, we next employed a Career Identity survey by using the Usefulness Construct ("students' beliefs in the usefulness of learning computer science") of Hoegh and Moskal's (2009) Attitudes Toward Computer Science instrument to understand students' future career aspirations in terms of computer science[3]. Again, overall the responses appeared to go down slightly from pre to post, however because of differences in sample sizes these data must be interpreted cautiously. Overall, most of the students agreed that computer science is very important for their careers – scoring somewhere between a 3 (Agree) and 4 (Strongly Agree) on a 4-point Likert-scale. A two-way ANOVA revealed that this agreement, unlike the previous instrument, did not differ by class or level ($p = .688$), nor did it differ from pre to post ($p = .981$).

Table 4: Total Sum Scores on Computer Career Identity Instrument by Course

	Pre		Post	
Course	n	$M(SD)$	n	$M(SD)$
CS 1	36	3.52 (0.44)	20	3.43 (0.55)
CS 2	19	3.59 (0.55)	10	3.64 (0.41)
CS Seminar	31	3.53 (0.47)	24	3.52 (0.44)
Capstone	18	3.48 (0.54)	11	3.53 (0.57)
Total	104	3.53 (0.48)	65	3.51 (0.49)

When examining these individual items at pre and post across all classes, it is important to note that three of the items were negatively worded, thus these items were reverse coded for the above overall average. For these items, lower scores actually indicate greater identity with pursuing computer science for a career. Basically all of the items experienced small declines from pre to post in identifying as a computer scientists as a career, except for *Knowledge of computing skills will not help me secure a good job* – for that item only more students thought that they needed computing skills from pre to post. None of the changes were statistically significant via independent samples t-tests.

Finally, more simply, we just asked, To what extent do you currently plan to pursue computer science in your career post-graduation? Overall, it seems that more students realize they will not pursue computer science across their courses, with the overall percent at pre of those that probably or definitely will pursue computer science 81% reducing to 73% at post, and the percent of those who definitely will not or probably will not pursue computer science as part of their career increasing from 8% at pre to 12% at post. At the same time however, the percent of students who definitely will pursue computers

Table 5: Individual Survey Item Mean Scores: Computer Career Identity Instrument

Course	Pre n=104 $M(SD)$	Post n=65 $M(SD)$
Developing computing skills will not play a role in helping me achieve my career goals.*	1.61 (0.98)	1.54 (0.92)
Knowledge of computing will allow me to secure a good job.	3.56 (0.55)	3.48 (0.62)
My career goals do not require that I learn computing skills.*	1.47 (0.67)†	1.60 (0.73)
Knowledge of computing skills will not help me secure a good job.*	1.51 (0.83)†	1.35 (0.54)
I expect that learning to use computing skills will help me achieve my career goals.	3.67 (0.51)	3.58 (0.68)

*Negatively worded and therefore reverse coded for overall average.
† Pre n=103

science solidifies, increasing from 44% at pre to 51% at post. Future work will investigate if this stabilizes or peaks in later courses.

Students were asked to elaborate through open-ended responses about identity:

- "It's who I am professionally and as a hobbyist, but it is not who I am."
- "I'm slowly learning that I don't really care about being a computer scientist and do not want to pursue that career path. I really don't consider myself an engineer."
- "I don't feel very confident in my abilities as a computer scientist or engineer."
- "I've been developing software for quite some time before I entered [the University of Portland], and would already partially identify as a computer scientist. However, working for a degree in CS has only solidified that identity."

The next steps of this project include continuing to collect these data and examine changes over time and over years.

4.3 What are the hinge-points of CS identity?

Across all pre and post responses, 61 included a written response that could be interpreted as a meaningful event that impacted the student's CS identity. The authors coded these roughly into the categories listed in Table 6. We further grouped these responses by **direct**: class(es) or work related to a class, **coding**-related, or **indirect**: declaring the major or an outside experience.

Table 6: Open response, stated hinge-points.

Direct	65.6%	**Coding**	18.0%	**Indirect**	16.4%
	n		n		n
Class(es)	28	Experience, from scratch, without help, or similar	11	Declaring the major	8
Critical thinking	4			Work or internship	2
Homework	4				
Group project	3				
Other (e.g. lingo)	1				

Although most educators would not be surprised by these breakdowns, it does help give some insight into where the curriculum might be changed in order to impact identity formation. In many cases we suspect that declaring the major stems from taking CS 1 and having a positive experience. A very small number of responses were clearly negative or mentioned the same cause as a both a positive and negative. "An impossible" assignment is one example of a negative from a student. We also want to point out that making **coding** a separate category emerged naturally, as many of these were also paired with statements of emerging confidence in other open text responses which could be a specific proxy for CS identity. In the future we will attempt to disentangle this effect.

5 Discussion and Future Work

This preliminary work shows hints of a formative change in CS 1, and shows that there is a significant change between that course and later CS courses.

This work is preliminary as we haven't yet run the survey long enough for a longitudinal analysis. The overall drop from pre to post might also contribute to an enthusiasm bias (based on attendance, willingness to complete rather than play on their phones, etc.) We also did not have enough Capstone respondents to deeply understand the year's graduating seniors' experiences.

Future work will include additional data collection, and a more detailed analysis of qualitative responses. Additionally, the authors think that more descriptive data could help more definitively answer where curricular interventions could have the most impact on developing a CS identity.

Acknowledgements

Thanks to the faculty that taught these courses for taking the time to help survey the students using class time and for their enthusiastic engagement.

References

[1] Sapna Cheryan et al. “Double isolation: Identity expression threat predicts greater gender disparities in computer science”. In: *Self and Identity* 19.4 (2020), pp. 412–434. DOI: 10.1080/15298868.2019.1609576. eprint: https://doi.org/10.1080/15298868.2019.1609576. URL: https://doi.org/10.1080/15298868.2019.1609576.

[2] Morgan M Fong et al. “Exploring Computing Students’ Sense of Belonging Before and After a Collaborative Learning Course”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1.* 2024, pp. 359–365.

[3] Andrew Hoegh and Barbara M Moskal. “Examining science and engineering students’ attitudes toward computer science”. In: *2009 39th IEEE frontiers in education conference.* IEEE. 2009, pp. 1–6.

[4] Amanpreet Kapoor and Christina Gardner-McCune. “Understanding CS Undergraduate Students’ Professional Identity through the lens of their Professional Development”. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education.* ITiCSE ’19. Aberdeen, Scotland Uk: Association for Computing Machinery, 2019, pp. 9–15. ISBN: 9781450368957. DOI: 10.1145/3304221.3319764. URL: https://doi.org/10.1145/3304221.3319764.

[5] Stephanie Lunn et al. “The Impact of Technical Interviews, and other Professional and Cultural Experiences on Students’ Computing Identity”. In: *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1.* ITiCSE ’21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 415–421. ISBN: 9781450382144. DOI: 10.1145/3430665.3456362. URL: https://doi.org/10.1145/3430665.3456362.

[6] Jonathan Mahadeo, Zahra Hazari, and Geoff Potvin. “Developing a computing identity framework: Understanding computer science and information technology career choice”. In: *ACM Transactions on Computing Education (TOCE)* 20.1 (2020), pp. 1–14.

[7] Sarah L Rodriguez and Kathleen Lehman. “Developing the next generation of diverse computer scientists: the need for enhanced, intersectional computing identity theory”. In: *Computer Science Education* 27.3-4 (2017), pp. 229–247.

Demographics, Approaches, and Conceptions: Understanding Computer Science Learning[*]

Bokai Yang[1], Ling Hao[2], Yuqi Song[3], Xin Zhang[3], Fei Zuo[4], Xianshan Qu[4]
[1]Department of Computer Science
University of Wisconsin – Eau Claire
Eau Claire, WI 54703
yangboka@uwec.edu
[2]School of Teaching & Learning
Illinois State University
Normal, IL 61790
lhao1@ilstu.edu
[3]Department of Computer Science
University of Southern Maine
Portland, ME 04101
{yuqi.song,xin.zhang}@maine.edu
[4]Department of Computer Science
University of Central Oklahoma
Edmond, OK 73034
{fzuo,xqu1}@uco.edu

Abstract

Understanding students' computer science learning conception and learning approach is essential for improving their learning experience and performance. By applying qualitative content analysis, machine learning

[*]Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

techniques and a descriptive approach, this study explores the associations between demographic characteristics, computer science learning approaches, and conceptions among undergraduate students. The findings of this study reveal associations between students' learning approaches and demographic factors such as grades, gender, and parents' education. Moreover, the study highlights discrepancies between students' learning approaches and their motivation and strategy for learning.

1 Introduction

In the rapidly evolving landscape of education, the intersection of technology and pedagogy has given rise to innovative methodologies for understanding and enhancing students' learning experiences. Computer science emerged as an essential discipline in the last century and its technique has been widely applied to enhance learning. However, as computer science educators, our understanding of students' computer science learning conceptions remains limited[8],[16].

Learning concepts refer to learner's beliefs about learning and how learner makes sense of learning through learning experiences[1], [11], [16]. Previous research supports the idea that students' learning conceptions are related to their learning approach and performance[1], [13]. In addition, many studies have shown that students' learning beliefs can be a strong predictor of their academic performance[8]. Students' conceptions are important in their learning processes and need to be considered in teaching. Understanding and addressing students' computer science learning conceptions can lead to more effective instructional strategies tailored to individual needs.

In education, there is a broad agreement on the importance of incorporating students' conceptions into teaching practices[2], [15], [12], [5]. The connection between students' learning conception and their approach to achieve it might persist unaltered unless there is intervention from the instructor[1]. Thus, to enhance students' learning experiences and outcomes, educators must be able to understand students' learning conceptions and access their learning conceptions.

In recent years, there has been an increasing interest in applying relevant educational research methods into computer science education. Researchers have been trying to understand and facilitate the learning experience of students[8], [14], [16], [7]. In the study of [8], the researchers explored how students' learning conception influences their computational thinking and programming self-efficacy. Other researchers in [16] applied a novel draw-a-picture technique to explore college students' computer science learning conception. Their results show differences in computer science learning conceptions across student characteristics. These findings underscore the importance of individual differences in learning conceptions when designing effective computer science curriculum and instructional strategies.

The integration of technology has given rise to innovative approaches aimed at unraveling the complexities of students' learning process. Learning analytics, with its ability to scrutinize datasets, offers an opportunity to unravel patterns, trends, and potential demographic factors that might impact computer science students' learning conceptions. With the goal of optimizing learning experience, learning analytics usually involves data analysis with the help of statistics and machine learning. A number of studies attempt to apply learning analytics to automatically identify, analyze or present students' learning behavior[3], [11], [17], [4]. By leveraging insights gleaned from learning analytics, educators can tailor their teaching methodologies to better align with the diverse needs and preferences of individual learners.

While researchers have invested significant effort in understanding and enhancing students' computer science learning experiences, only a limited number of studies have explored students' conceptions of computer science learning. Even less research analyzed the associations between student characteristics and their learning conception. Computer science teachers continue to face the challenge of elucidating students' learning processes while respecting their individual learning conceptions[12]. Since students' learning conceptions are considered to be important to their learning approach and performance[1], [13], it is critical to help computer science instructors and facilitators to understand and access students' computer science learning conceptions. The objective of this study is to identify students' computer science learning conception patterns and explore the association between students' demographic and their learning conceptions in computers science courses. We aim to answer the following research questions:

Q1 What are the patterns of computer science learning conceptions observed among the college students who participated in this research?

Q2 How do specific demographic characteristics align with each computer science learning conception pattern?

The rest of the paper is organized as follows: Section 2 describes the data collection method and the data analysis process. Section 3 presents the patterns of students' computer science learning conception, the data analysis results and discusses the associations between students' learning conceptions and demographic information. Section 4 discusses the limitations of the current study and provides insights for future research. Section 5 concludes the paper.

2 Methodology

This study applied qualitative content analysis and a descriptive approach. A 49-question survey was distributed to over 150 students majored in computer science at a public comprehensive university located in the middle north part of the United States. Thirty-six students chose to participate in the research project. Among the students, 15 were female and 21 were male (with other gender identity options provided). The age range of the participating students is 18-24. All students who participated in this study took courses related to learning computer science. The participating students are at various stages of their academic studies: 39% freshmen, 19% sophomores, 31% juniors, and 11% seniors. The racial composition of the participating students is as follows: White (72%), Black (6%), Asian (16%), American Indian (3%), and of Brazilian or Latino descent (3%). The university's IRB approved the study, and the students voluntarily participated in the data collection process.

The survey consisted of three sections. The first section includes 9 questions related to students' demographic data. The second section provides 7 questions about their previous programming experiences and learning preferences. For the first two sections, the option 'prefer not to answer' is provided to allow the students to opt out of answering any question. The third section asks questions about students' conceptions of learning computer science and approaches to learning computer science. The questions in the third section are adopted from the survey questions of [14]. In section three, the 26 questions related to students' conceptions of learning computer science are originally designed by [7] and further adapted and renamed to COLCS by [10]. This survey was first developed to investigate students' conceptions of learning computer science by [7]. The questions cover factors including memorizing (three items), testing (four items), calculating and practicing (three items), programming (four items), increasing one's knowledge (three items), application and understanding (four items), and seeing in a new way (five items). The other 17 questions in section three are used to measure students' approaches to learning computer science; the original version of the questions was designed by [6] and revised by [9]. The factors for approaches include surface motive (three items), surface strategy (four items), deep motive (six items), and deep strategy (four items). All the questions in section three use a 5-point scale measurement system. The questions in other sections are multiple choice questions.

The survey data were extracted into 2 parts which include the demographic part (section 1 and section 2) and CSCA (computer science conceptions and approaches) part (section 3). The CSCA data were analyzed using the K-Means Clustering method as a technique for performing data groupings. This analysis was performed using Weka Software version 3.6.3. The optimal number of clusters is decided by Euclidean Distance of each approach and the Elbow method. Based on the K-Means Clustering approach in the previous step, the optimal clusters will be obtained. The optimal clusters are further analyzed with the demographic data to summarize the demographic characteristics.

3 Results and Discussion

Based on the descriptive K-Means analysis, as shown in Figure 1 and Figure 2, we found 5 clusters with different cluster characteristics.

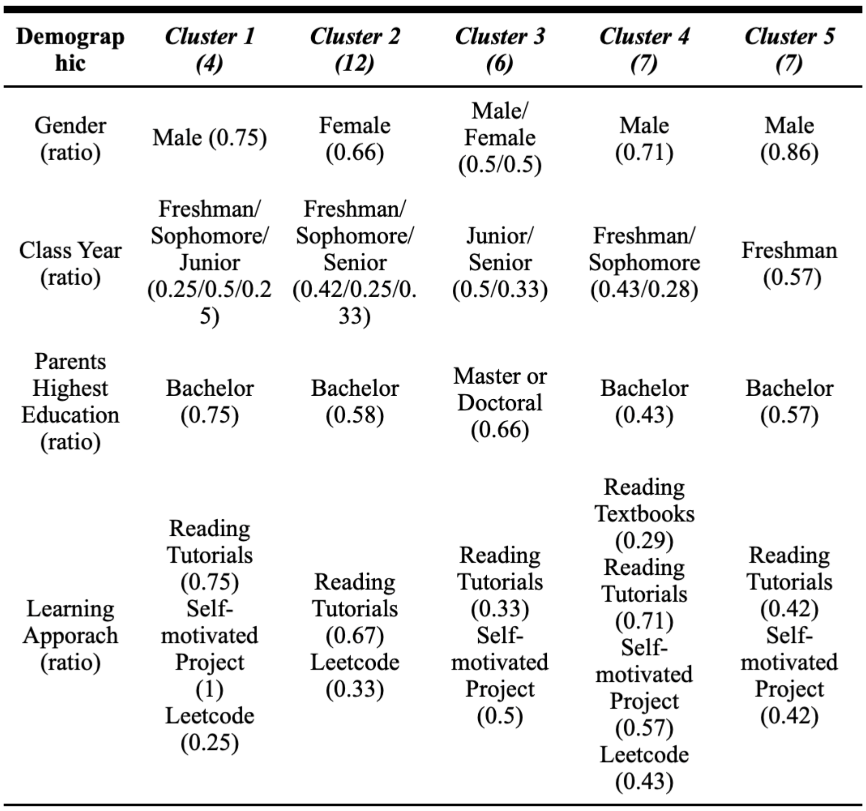
Figure 1 describes the demographic characteristics of each cluster. After analyzing the connections between demographic factors and clusters, we discovered that gender, class year, and parents' highest education level are three factors associated with the clusters. Cluster 2 tends to have more female students, while other clusters are dominated by male students. From the perspective of class year, almost all students in cluster 3 are juniors and seniors. On the other hand, students in lower grade levels are concentrated in clusters 4 and 5. Additionally, parents of students in cluster 3 have higher education levels compared to those in other clusters. When asked about their common computer science learning approach, students in cluster 2 tend to learn computer science through tutorials instead of coding practice. The students in cluster 3 and cluster 5 exhibit a preference for hands-on coding practice over tutorial-based learning approaches.

On the contrary, academic performance in high school, SAT/ACT score, and current GPA do not appear to be significant influential factors on computer science students' learning conception and approach. All clusters tend to exhibit similar distributions of these factors. Although previous studies show that a strong background in mathematics is associated with computer science students' performance[3], the mathematics background appears to be a less significant factor to students' computer science learning conception and approach in this study.

The clustering result and the selected attributes of each cluster are shown in Figure 2. We listed the attributes that reflect the characteristics of the clusters in Figure 2.

Cluster 1 comprises 4 students, of whom 3 are male, covering three grades. We observed that students in cluster 1 generally agree with all the question statements in section 3, except for those related to surface motivation and surface strategy. Conversely, they strongly agree with the statements regarding deep motivation and deep strategy. Additionally, analysis of the qualitative responses provided by students in Cluster 1 revealed common preferences for self-motivated projects.

The 12 students in cluster 2 tend to agree with all the statements in section 3 except the statements for surface motivation and surface strategy. A noteworthy point is that 66% of the students in cluster 2 are female students. As shown in the cluster characteristic result, although students in cluster 2 disagree with the surface strategy statements, they rely more on textbooks and reading materials compared to coding practice and projects. This suggests a potential divergence in learning preferences between genders within this clus-



Demographic	***Cluster 1 (4)***	***Cluster 2 (12)***	***Cluster 3 (6)***	***Cluster 4 (7)***	***Cluster 5 (7)***
Gender (ratio)	Male (0.75)	Female (0.66)	Male/ Female (0.5/0.5)	Male (0.71)	Male (0.86)
Class Year (ratio)	Freshman/ Sophomore/ Junior (0.25/0.5/0.25)	Freshman/ Sophomore/ Senior (0.42/0.25/0.33)	Junior/ Senior (0.5/0.33)	Freshman/ Sophomore (0.43/0.28)	Freshman (0.57)
Parents Highest Education (ratio)	Bachelor (0.75)	Bachelor (0.58)	Master or Doctoral (0.66)	Bachelor (0.43)	Bachelor (0.57)
Learning Apporach (ratio)	Reading Tutorials (0.75) Self-motivated Project (1) Leetcode (0.25)	Reading Tutorials (0.67) Leetcode (0.33)	Reading Tutorials (0.33) Self-motivated Project (0.5)	Reading Textbooks (0.29) Reading Tutorials (0.71) Self-motivated Project (0.57) Leetcode (0.43)	Reading Tutorials (0.42) Self-motivated Project (0.42)

Figure 1: Clusters (size) and Selected Demographic

ter, underscoring the need for further investigation into the underlying factors leading to these differences.

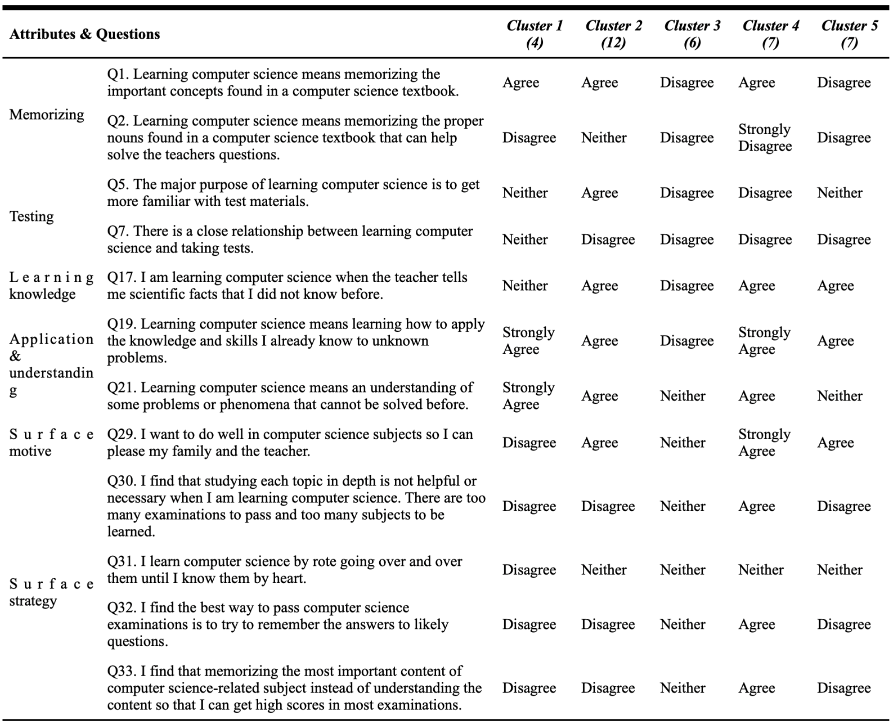
Cluster 3 has 6 students. The demographic data shows that almost all the students in cluster 3 are junior and senior students, and their parents tend to have higher levels of education. The clustering result shows that these students disagree with some statements which others agree with. Such statements include Q1 for memorizing, Q5 for testing, Q17 for learning knowledge, and Q19 and Q21 for application and understanding. In addition, the students in cluster 3 do not express positive or negative opinions on the surface motivation and surface strategy statements. Although not shown in Figure 2, it's noteworthy that the students in cluster 3 agree with all the statements related to deep motivation and deep strategy.

Most of the students in cluster 4 are male students (71%) with 43% freshmen and 28% sophomores. These students disagree with the approach of memorizing proper nouns from textbooks and preparing for exams based on test materials. However, they agree with the surface motivation and surface strategy approaches. Additionally, analysis of their responses indicated a preference for tutorial-based learning approaches. This indicates an underlying preference within this cluster for passive learning strategies, highlighting a potential need for interventions aimed at promoting more active engagement and participation in the learning process.

There are 7 students in cluster 5 and most of the students are male (86%). About half of the students in this cluster are freshmen (57%). These students disagree with the statement that learning computer science is to memorize the important concepts and do not apply a surface strategy. They prefer both practical, hands-on learning experiences and tutorial-based learning approaches.

From our clustering result, we can see that students hold different views on test materials, which may lead to different perspectives on surface strategy. While learning knowledge and application and understanding are highly related to students' deep strategy, memorizing and testing were found to be positive predictors of surface strategy but negative predictors for deep strategy[16]. However, this pattern does not hold true for students in cluster 3 and cluster 4. Students' computer science learning approaches may not be consistent with their learning motivation and strategy. The associations between approaches, motivation, and strategy may vary for students at different phases of their academic journey. In addition, students hold different views on Q5 and have different learning approaches. Our findings suggest that undergraduate students have varying perspectives on how they conceive learning computer science. Instructors should take this into consideration when assessing students' learning outcomes.

Students who agree with surface motivation and surface strategy may be



Attributes & Questions		*Cluster 1 (4)*	*Cluster 2 (12)*	*Cluster 3 (6)*	*Cluster 4 (7)*	*Cluster 5 (7)*
Memorizing	Q1. Learning computer science means memorizing the important concepts found in a computer science textbook.	Agree	Agree	Disagree	Agree	Disagree
	Q2. Learning computer science means memorizing the proper nouns found in a computer science textbook that can help solve the teachers questions.	Disagree	Neither	Disagree	Strongly Disagree	Disagree
Testing	Q5. The major purpose of learning computer science is to get more familiar with test materials.	Neither	Agree	Disagree	Disagree	Neither
	Q7. There is a close relationship between learning computer science and taking tests.	Neither	Disagree	Disagree	Disagree	Disagree
Learning knowledge	Q17. I am learning computer science when the teacher tells me scientific facts that I did not know before.	Neither	Agree	Disagree	Agree	Agree
Application & understandin g	Q19. Learning computer science means learning how to apply the knowledge and skills I already know to unknown problems.	Strongly Agree	Agree	Disagree	Strongly Agree	Agree
	Q21. Learning computer science means an understanding of some problems or phenomena that cannot be solved before.	Strongly Agree	Agree	Neither	Agree	Neither
Surface motive	Q29. I want to do well in computer science subjects so I can please my family and the teacher.	Disagree	Agree	Neither	Strongly Agree	Agree
Surface strategy	Q30. I find that studying each topic in depth is not helpful or necessary when I am learning computer science. There are too many examinations to pass and too many subjects to be learned.	Disagree	Disagree	Neither	Agree	Disagree
	Q31. I learn computer science by rote going over and over them until I know them by heart.	Disagree	Neither	Neither	Neither	Neither
	Q32. I find the best way to pass computer science examinations is to try to remember the answers to likely questions.	Disagree	Disagree	Neither	Agree	Disagree
	Q33. I find that memorizing the most important content of computer science-related subject instead of understanding the content so that I can get high scores in most examinations.	Disagree	Disagree	Neither	Agree	Disagree

Figure 2: Clusters (size) and Selected Attributes

more motivated by passing exams, getting higher grades or meeting teacher/-family expectations. They tend to remember rote information to pass exams. On the other hand, students who agree with deep motivation and deep strategy appear to be motivated by interest and curiosity. These students tend to make meaning of the course contents and connect prior knowledge with the new topics they learnt. According to our findings, students in almost all the clusters strongly agree with surface motivation and disagree with the surface strategy except for the students in cluster 4. This result is consistent with the result of [16] which indicates that students surface motivation drives students' behavior more than deep motivation.

4 Limitation and Future Research

This study has some limitations. First, the sample size of the student participants is limited, and all the student participants are from one university in the middle north of United States. The result may not be able to represent all undergraduate students. Second, we only include self-reported academic performance as the factors. It is possible that the self-reported academic performance may not reflect students' real performance.

For future work, more data from different universities across different regions could be collected. This would facilitate a broader representation of computer science students, thereby enriching the findings. In addition, statistical analysis could be applied to further analyze the relationship between the demographic factors and learning conceptions and approaches clusters. We also noticed a potential association between race and the use of surface strategy that could be further investigated. Moreover, incorporating qualitative methods such as interviews or focus groups could provide deeper insights into students' perceptions and experiences, complementing the quantitative data collected in this study.

5 Conclusion

In conclusion, this study sheds light on how undergraduate computer science students' demographic characteristics associate with their computer science learning approaches and conceptions. Students' learning approaches are associated with their grades, gender and parents' education. Additionally, students' learning approaches may not match with their learning motivation and strategy. Our findings underscore the importance of understanding these differences in order to tailor instructional strategies effectively. Future research should aim to address these limitations by collecting data from a more diverse range of universities and utilizing a combination of quantitative and qualitative methods. By doing so, we can gain a deeper understanding of the factors influencing students' learning experiences and better support their academic success in computer science education.

References

[1] MA Alkhateeb and OAQB Milhem. *Student's concepts of and approaches to learning and the relationships between them. Cakrawala Pendidikan, 39 (3), 620–632.* 2020.

[2] Tamer G Amin, Carol L Smith, and Marianne Wiser. "Student conceptions and conceptual change: Three overlapping phases of research". In: *Handbook of Research on Science Education, Volume II.* Routledge, 2014, pp. 57–81.

[3] Heidi Burgiel, Philip M Sadler, and Gerhard Sonnert. "The association of high school computer science content and pedagogy with students' success in college computer science". In: *ACM Transactions on Computing Education (TOCE)* 20.2 (2020), pp. 1–21.

[4] Heeryung Choi et al. "Logs or self-reports? Misalignment between behavioral trace data and surveys when modeling learner achievement goal orientation". In: *LAK23: 13th international learning analytics and knowledge conference.* 2023, pp. 11–21.

[5] AA diSessa. "A history of conceptual change research: Threads and fault lines. comparisons". In: *Science Education* 99.3 (2014), pp. 410–416.

[6] David Kember, John Biggs, and Doris YP Leung. "Examining the multidimensionality of approaches to learning through the development of a revised version of the Learning Process Questionnaire". In: *British Journal of Educational Psychology* 74.2 (2004), pp. 261–279.

[7] Min-Hsien Lee, Robert E Johanson, and Chin-Chung Tsai. "Exploring Taiwanese high school students' conceptions of and approaches to learning science through a structural equation modeling analysis". In: *Science Education* 92.2 (2008), pp. 191–220.

[8] Silvia Wen-Yu Lee et al. "Students' beliefs about computer programming predict their computational thinking and computer programming self-efficacy". In: *Interactive Learning Environments* (2023), pp. 1–21.

[9] Jyh-Chong Liang, LEE Min-Hsien, and TSAI Chin-Chung. "The Relations Between Scientific Epistemological Beliefs and Approaches to Learning Science Among Science-Major Undergraduates in Taiwan." In: *Asia-Pacific Education Researcher (De La Salle University Manila)* 19.1 (2010).

[10] Jyh-Chong Liang, Yi-Ching Su, and Chin-Chung Tsai. “The assessment of Taiwanese college students’ conceptions of and approaches to learning computer science and their relationships”. In: *The Asia-Pacific Education Researcher* 24 (2015), pp. 557–567.

[11] William G Perry Jr. *Forms of Intellectual and Ethical Development in the College Years: a Scheme. Jossey-Bass Higher and Adult Education Series.* ERIC, 1999.

[12] Judith Stanja et al. “Formative assessment strategies for students’ conceptions—The potential of learning analytics”. In: *British Journal of Educational Technology* 54.1 (2023), pp. 58–75.

[13] Imam Suyitno et al. “How prior knowledge, prospect, and learning behaviour determine learning outcomes of BIPA students?” In: *Jurnal Cakrawala Pendidikan* 38.3 (2019), pp. 499–510.

[14] Karthikeyan Umapathy, Albert D Ritzhaupt, and Zhen Xu. “College students’ conceptions of learning of and approaches to learning computer science”. In: *Journal of Educational Computing Research* 58.3 (2020), pp. 662–686.

[15] Stella Vosniadou et al. *International handbook of research on conceptual change*. Vol. 259. Routledge New York, 2008.

[16] Zhen Xu et al. “Exploring college students’ conceptions of learning computer science: A draw-a-picture technique study”. In: *Computer Science Education* 31.1 (2021), pp. 60–82.

[17] Bokai Yang et al. “Untangling chaos in discussion forums: A temporal analysis of topic-relevant forum posts in MOOCs”. In: *Computers & Education* 178 (2022), p. 104402.

Large Language Model-Supported Software Testing with the CS Matrix Taxonomy*

Johannah L. Crandall[1] and Aaron S. Crandall[2]
[1]Department of Mathematics
Spokane Falls Community College
Spokane, WA 99224
Johannah.Crandall@sfcc.spokane.edu
[2]Department of Computer Science
Gonzaga University
Spokane, WA 99258
crandall@gonzaga.edu

Abstract

New breakthroughs in code synthesis from Generative Pre-Trained Transformers (GPT) and Large Language Model (LLM) algorithms are driving significant changes to software engineering education. Having algorithms able to generate components of a software project means that software developers will need stronger skills in requirements specification to guide code generation as well as stronger skills in code review, testing, and integration to incorporate AI-generated code into projects. Shifts in industry and classroom practices are already occurring with the availability of inline code generation tools like GitHub's Copilot, which makes discussion of pedagogical strategies in this area a timely topic. Of immediate concern in computer science education is the potential for LLM-generated code and code help to undermine the learning of CS students. In order to avoid such undermining in even intentional uses of LLM-enhanced learning supports, it is necessary to clarify the roles such supports need to play in the pedagogical process. The Computer Science

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Matrix Taxonomy provides a strong framework for organizing software testing learning outcomes as well as delineating the operational space in which LLM-based feedback tools should operate to support those learning outcomes. In this paper, the authors operationalize the CS Matrix Taxonomy for software testing learning outcomes and illustrate the integration of LLM-generated test strategy suggestions as an extension of the peer coding/testing model. The work includes examples of AI-generated code testing suggestions that students would use to help guide their own code synthesis for assignments or projects.

1 Introduction

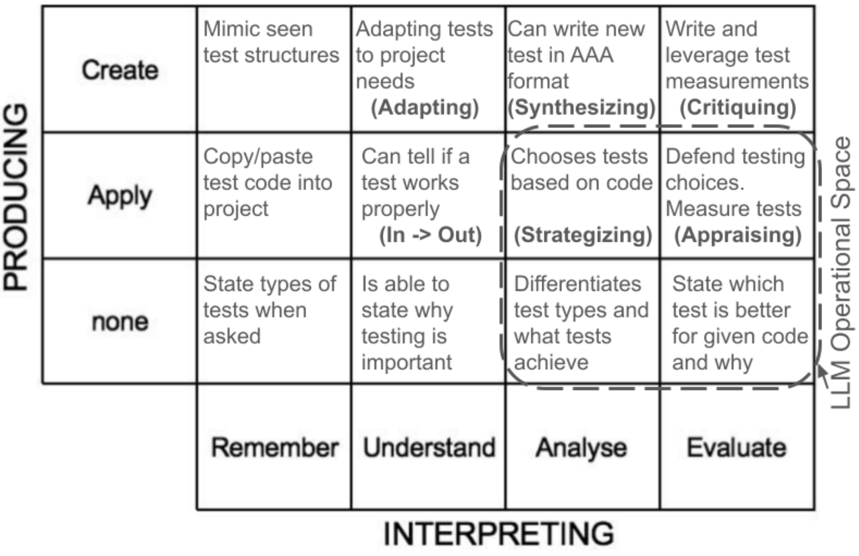
Software testing has become a keystone of modern software development [8], and students learning to be software engineers should have a solid set of fundamental software testing skills. Historically, software testing was viewed as an activity done after a project was complete, or as only part of a bugfixing effort. Today, testing is seen as best done either at the same time as the code development, or even better through Test Driven Development (TDD) where tests are written first [1].

To teach students how to write tests, learn their importance [2], and to gain an understanding of how tests integrate with a project is a significant pedagogical effort [10, 14]. Garousi et al. [6] summarized the field of software-testing education where several of their conclusions revealed how "spotty" or intermittent software testing education remains within curricula. Some of the reasons for this lack in the classroom is often the series of learning steps new programming students have to go through before they are able to implement software tests in code. The students need to have a reasonably functional level of understanding of how code is written, how data is stored in computing systems, and have the ability to write functions that are testable. This means that students need several skills before they are able to start writing tests in the first place.

Once students can start writing tests, introducing testing to their skill set presents a scaling problem. Code reviewing and code feedback is a work intensive effort [9]. This work is often abbreviated by faculty and teaching assistants unless significant resources are spent to hire enough qualified code reviewers. Alternative routes to code feedback include peer reviewing [7, 15] or some kind of automated code feedback system [3, 11, 12]. Translating this problem to teaching software testing skills presents similar problems of scale and complexity of automation.

To help automate software testing education, several approaches have been tried in the past. For example, Zheng et al. [16] worked to use a combination of Natural Language Processing and information retrieval to scale software testing suggestion systems to students. These kinds of tools have shown some success, but prior to Large Language Models (LLM) like ChatGPT or Code Llama [13] the systems were very limited in what kinds of testing suggestions they could give. With deep AI models, novel and more complete solutions to AI-generated and AI-driven teaching models are now possible and should be explored.

In order to lay the groundwork for assessing LLM-enhanced teaching methods as well as student learning outcomes using LLM-supported methods, this paper harnesses the two-dimensional adaptation of Bloom's Taxonomy by Fuller et al. [5] called the CS Matrix Taxonomy. We operationalize the CS Matrix Taxonomy specifically for software testing learning outcomes, which is discussed in Section 2. To demonstrate how this LLM-based test suggestion system works in the context of a student learner, the effects of integrating it into the coding workflow is outlined in Section 3. The LLM-enhanced feedback approach built by the authors does not generate the test itself in code form, but instead generates a description of how the software could be tested, as shown in Section 4. Overall, this approach supports their traversal through the CS Matrix Taxonomy on both conceptual and procedural axes, fostering stronger conceptual understanding than the limited feedback they currently receive or possible dependency on code completely generated by AI models.


PRODUCING
INTERPRETING
LLM Operational Space

Create	Mimic seen test structures	Adapting tests to project needs (Adapting)	Can write new test in AAA format (Synthesizing)	Write and leverage test measurements (Critiquing)
Apply	Copy/paste test code into project	Can tell if a test works properly (In -> Out)	Chooses tests based on code (Strategizing)	Defend testing choices. Measure tests (Appraising)
none	State types of tests when asked	Is able to state why testing is important	Differentiates test types and what tests achieve	State which test is better for given code and why
	Remember	Understand	Analyse	Evaluate

Figure 1: The CS Matrix Taxonomy model applied to software testing education.

2 Operationalizing the CS Matrix Taxonomy for Software Testing Learning Outcomes

In a normal software development course at the CS2 or CS3 levels, the concept of the unit test is often introduced. The students are likely shown the Arrange-Act-Assert (AAA) pattern for developing tests [4]. They likely receive both readings and examples on why testing is valuable and how it can be done on small projects. From there, they ideally have in-class or in-lab experiences writing tests on example code before applying it to their own projects. Various interacting stages of conceptual and procedural competency during this process are shown using the Computer Science Matrix Taxonomy from Fuller et al.'s work [5] in Figure 1.

The CS Matrix Taxonomy's horizontal Interpreting axis follows the progression of conceptual/analytical sophistication while the vertical Producing axis illustrates implementation and extension at each of those conceptual levels. In terms of software testing, movement along these two axes together, from the bottom left to the upper right, comprises the transition from rote mimicry of prescribed tests to a depth of understanding that facilitates intentional and strategic test generation.

As CS learners move from left to right on the Interpreting axis in their software testing work, they progress from simply replicating provided tests, to adapting tests to new projects, to generating entirely new testing structures in context. This developmental path involves them remembering how tests work, mapping the suggestions onto their own code, then understanding the suggestions, determining how they are built, then evaluating their value in demonstrating whether their own code is implemented properly.

A learner's progression through the Matrix is benefited by timely and sufficient feedback of appropriate depth and diversity, which is difficult to get from graders in large classes. An LLM-enhanced peer-style feedback approach could help the learner progress through the learning outcomes by providing test strategy suggestions customized to the student's work. Testing suggestions provided to the student are generated within the LLM Operational Space illustrated in Figure 1. The LLM-enhanced feedback to the student is designed to elicit and further student understanding through test suggestions and partial examples, without providing explicit code solutions. This is designed to keep the LLM output confined to the *Strategizing* and *Appraising* activities, which are the primary work done by peer reviewers or teaching assistants. The result is that the student remains responsible for *Synthesizing* and *Critiquing* activities from the Matrix Taxonomy.

3 Integration of LLM-Enhanced Feedback into Student Software Engineering & Testing Pattern

Software testing ideally occurs as an integral part of a complex software development process which poses many challenges for the learner. Students require the ability to read, parse, and understand project requirements to design a software solution, write the code to implement their ideas, and generate a testing strategy for the code they have written. This activity workflow is often introduced in second (CS2) or third (CS3) semester Computer Science classes.

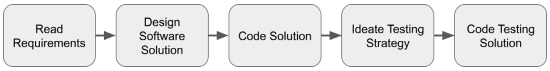


Figure 2: Expected early student learner software testing workflow.

A visualization of the expected student learner workflow is shown in Figure 2. At every stage of this workflow, there are opportunities for students to need help in understanding concepts and executing skills needed to generate a possible approach to solving the stage. To continue through the workflow to software testing, students are often asked to understand how code works from both algorithmic and data input/output perspectives to ideate a software testing strategy for their own code.

When starting to produce unit tests for their own projects, learners are faced with executing several simultaneous skills to make this happen. Examples of these skills include, but are not limited to:

1. Understanding their own code and the expected outcomes.
2. A strong grasp of the types of data for input and output.
3. Seeing where side effects might be caught and tested.
4. Looking at the different categories of possible unit testing to pick one to test for, including at least:
   (a) Expected positive cases
   (b) Negative cases
   (c) Exception handling
   (d) File and data handling
   (e) User input issues
   (f) Branch testing
5. Writing the code for the test in the AAA pattern.
6. Deciding if a given test they can think of is worth trying.

At the software testing stage, many students could use assistance to create a successful strategy before they try coding it. This is where peer learning strategies or instructor feedback are very helpful and where this paper's proposed LLM-enhanced testing suggestions fit into the student's workflow.

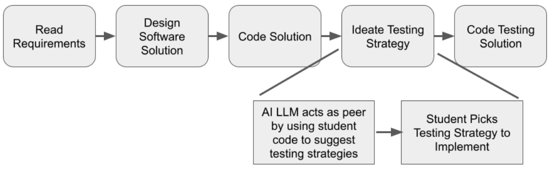

Read Requirements
Design Software Solution
Code Solution
Ideate Testing Strategy
Code Testing Solution
AI LLM acts as peer by using student code to suggest testing strategies
Student Picks Testing Strategy to Implement

Figure 3: AI-based LLM tool assisting in software testing strategy ideation.

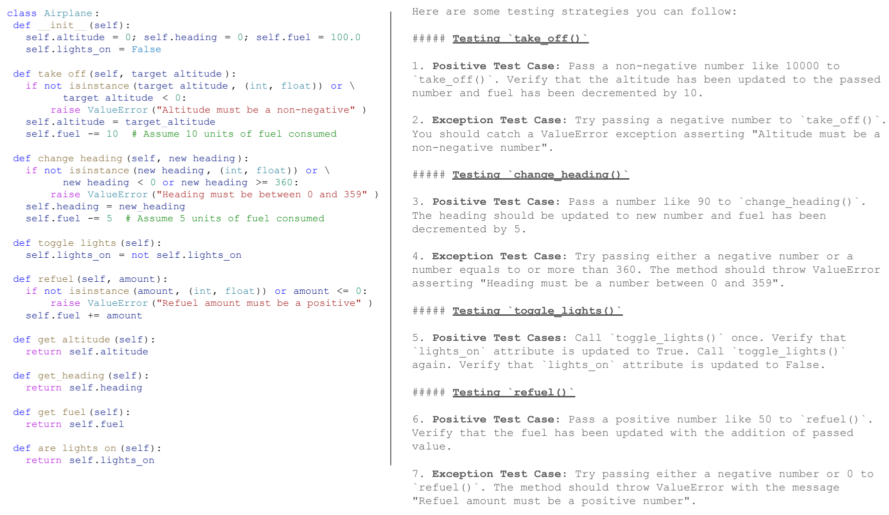
Students often need to be given a description of how to test a given project so they continue to develop their conceptual understanding of what to test and why, not just be given possible working tests for them to emulate. When a student desires help with their testing strategy ideation, the LLM-based tool would be able to become part of their workflow as shown in Figure 3. It is able to read their project, inspect their current tests, and suggest new approaches to testing the code which gives examples of testing approaches, which is a *Strategizing* activity, as well as feedback about the quality of the student's current tests, which is an *Appraising* activity.

As shown in Section 4, the tool writes the suggestions as English prose, which necessitates the student carrying out the *Adaptation*, *Synthesizing*, and *Critiquing* activities from the CS Matrix Taxonomy in Figure 1. The effort of synthesizing the testing code themselves serves to both integrate their coding skills as well as their ability to understand the computing algorithm implemented in their code itself. Good testing suggestions will also help them see how the tests tie back to the project requirements, which will be an invaluable stage in changing from post-hoc testing workflows that we see in Figure 2 and Figure 3 to then move towards a TDD approach as their skills mature, allowing them to develop tests as a design specification.

4 Example of a LLM-Based Tool's Testing Suggestions

The proposed tool from Section 3 uses the student's code source files and related test code files to generate suggestions for further testing. To illustrate how this tool works, Figure 4 shows both a simple class that implements an airplane simulation along with the output of the LLM-based tool. The tool was prompted to read the code and suggest a variety of tests that could be implemented for the given source code.

Note that the suggested tests include descriptions of what categories of unit tests they are. These include positive, negative, and exception tests in this case. The categories of testing could be expanded or shrunk as needed.



```python
class Airplane:
 def __init__(self):
   self.altitude = 0; self.heading = 0; self.fuel = 100.0
   self.lights_on = False

 def take_off(self, target_altitude):
   if not isinstance(target_altitude, (int, float)) or \
         target_altitude < 0:
       raise ValueError("Altitude must be a non-negative" )
   self.altitude = target_altitude
   self.fuel -= 10  # Assume 10 units of fuel consumed

 def change_heading(self, new_heading):
   if not isinstance(new_heading, (int, float)) or \
         new_heading < 0 or new_heading >= 360:
       raise ValueError("Heading must be between 0 and 359" )
   self.heading = new_heading
   self.fuel -= 5  # Assume 5 units of fuel consumed

 def toggle_lights(self):
   self.lights_on = not self.lights_on

 def refuel(self, amount):
   if not isinstance(amount, (int, float)) or amount <= 0:
       raise ValueError("Refuel amount must be a positive" )
   self.fuel += amount

 def get_altitude(self):
   return self.altitude

 def get_heading(self):
   return self.heading

 def get_fuel(self):
   return self.fuel

 def are_lights_on(self):
   return self.lights_on
```

```
Here are some testing strategies you can follow:

##### Testing `take_off()`

1. Positive Test Case: Pass a non-negative number like 10000 to
`take_off()`. Verify that the altitude has been updated to the passed
number and fuel has been decremented by 10.

2. Exception Test Case: Try passing a negative number to `take_off()`.
You should catch a ValueError exception asserting "Altitude must be a
non-negative number".

##### Testing `change_heading()`

3. Positive Test Case: Pass a number like 90 to `change_heading()`.
The heading should be updated to new number and fuel has been
decremented by 5.

4. Exception Test Case: Try passing either a negative number or a
number equals to or more than 360. The method should throw ValueError
asserting "Heading must be a number between 0 and 359".

##### Testing `toggle_lights()`

5. Positive Test Cases: Call `toggle_lights()` once. Verify that
`lights_on` attribute is updated to True. Call `toggle_lights()`
again. Verify that `lights_on` attribute is updated to False.

##### Testing `refuel()`

6. Positive Test Case: Pass a positive number like 50 to `refuel()`.
Verify that the fuel has been updated with the addition of passed
value.

7. Exception Test Case: Try passing either a negative number or 0 to
`refuel()`. The method should throw ValueError with the message
"Refuel amount must be a positive number".
```

Figure 4: Example of LLM-generated test suggestions for a simple class.

Running the tool and generating this set of seven unit test suggestions takes around 45–90 seconds. For larger projects or on self-hosted LLM services like Ollama, it could take several minutes, which is still significantly faster than TA-derived feedback which could take hours or days to deliver. While the suggestions are not in-depth testing strategies with code of this complexity, for new software testing learners, it presents the core philosophies and approaches they should use for testing their code. Additionally, it is based on their specific code, instead of being a general description of the approach or an example presented on other code as they see in lectures and reading materials, which makes this an AI-driven personalized, tailored output akin to a peer or teaching assistant giving suggestions after reading the learner's code and giving suggestions.

5 Summary

Software testing and testing skills hold pivotal roles in modern software development. Students face challenges in learning testing due to its intermittent inclusion in curricula and the prerequisite skills needed before delving into test writing. Software testing is especially valuable to establish early in a degree program.

The goal of Bloom's and the CS Matrix Taxonomy is for learners to traverse learning stages and for that traversal to be evident to educators. For students in CS2 and CS3 courses, peer evaluators and course graders should be providing feedback to give ideas about what *Strategies* to test with and *Appraisals* of the student's existing work. This feedback is designed to help the learner *Synthesize* and *Critique* new tests, which demonstrates a high level of competency with the subject matter.

This work harnesses the CS Matrix Taxonomy to identify learning outcomes in software testing on both conceptual and procedural axes, then proposes integrating Large Language Models (LLMs), such as Code Llama or ChatGPT, into software testing education to facilitate learner progression through the taxonomy. The authors have developed such an LLM-enhanced tool that would generate descriptions of how the software could be tested, fostering conceptual understanding of code behavior without generating explicit test code to the learner. The tool provides strong hints about test categories, data input/output, and areas needing testing, offering students guidance similar to that provided by peers or instructors.

This paper also outlines the proposed tool's integration into students' coding workflow, as well as examples of testing strategy suggestions for simple projects. By harnessing a system capable of providing timely and effective feedback, this approach should improve the student learning outcomes including the understanding of unit testing concepts and enhanced skills in executing simultaneous tasks required for effective testing.

6 Future Work and Ideas

Future work for this project should flesh out the research on several axes. Notably, it should finish integrating the code testing suggestion tool into various workflow systems such as GitHub Actions, VS Code or other IDEs, and as a standalone tool to read files on an ad-hoc basis. All of these implementations would make the tool more accessible to computer science students and industry users to help them with simple testing strategy suggestions.

The tool's output and feedback should be evaluated for use in the classroom, similarly to other projects using GPT generated code reviews [3, 12]. This includes research projects evaluating the breadth, depth, and accuracy of the suggested tests. It should have the parameters tuned to give specific suggestions and a slate of parameters published to allow teachers to choose what kinds of suggestions the tool would focus on for their students. A series of in-class evaluations would be needed to evaluate which kinds and styles of suggestions are most effective at improving student skill outcomes.

An example future research project would be to develop metrics and evaluate the test suggestions before testing it with student groups. The study would be centered around generating and reviewing the test suggestions themselves. This could be done by gathering a corpus of student project code from the target CS2 and CS3 courses. Then generating test suggestions on the project code. The suggestions would be scored for validity, accuracy, coverage, prose clarity, and terminology use. Metrics for these scores is a novel field of study and would be a required step in developing these kinds of systems. After metrics and categories of evaluation are developed, then the evaluation process with the same student code corpus can be used across different LLMs to compare and contrast their abilities to select or train them as needed for use in the field.

The use of LLM/GPT models to parse and generate code is impressive and in its infancy. Leveraging these tools to help future generations of coders will be invaluable to improving student outcomes going forward.

Acknowledgements

Work generously supported by the R.W. Gillette Challenge Fund, the McDonald Student Work Fund, and the Gonzaga University Bollier Center for Integrated Science & Engineering.

References

[1] Samar Alsaqqa, Samer Sawalha, and Heba Abdel-Nabi. "Agile Software Development: Methodologies and Trends". In: *International Journal of Interactive Mobile Technologies* 14.11 (2020), pp. 246–270. DOI: 10.3991/ijim.v14i11.13269.

[2] Alberto Bacchelli and Christian Bird. "Expectations, outcomes, and challenges of modern code review". In: *International Conference on Software Engineering.* ICSE. IEEE. 2013, pp. 712–721. DOI: 10.1109/ICSE.2013.6606617.

[3] Aaron S. Crandall, Gina Sprint, and Bryan Fischer. "Generative Pre-Trained Transformer (GPT) Models as a Code Review Feedback Tool in Computer Science Programs". In: *The Journal of Computing Sciences in Colleges* 39.1 (Oct. 2023), pp. 38–47. ISSN: 1937-4771.

[4] Erik Dietrich. *Starting to Unit Test: Not as Hard as You Think.* BlogIntoBook.com, 2014.

[5] Ursula Fuller et al. "Developing a computer science-specific learning taxonomy". In: *ACM SIGCSE Bulletin* 39.4 (2007), pp. 152–170. DOI: 10.1145/1345375.1345438.

[6] Vahid Garousi et al. “Software-testing education: A systematic literature mapping”. In: *Journal of Systems and Software* 165 (2020), p. 110570. DOI: 10.1016/j.jss.2020.110570.

[7] Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. “A review of peer code review in higher education”. In: *ACM Transactions on Computing Education* 20.3 (Sept. 2020), pp. 1–25. DOI: 10.1145/3403935.

[8] Kainat Khan and Sachin Yadav. “A Literature Review on Software Testing Techniques”. In: *Optimization of Automated Software Testing Using Meta-Heuristic Techniques*. Springer International Publishing, 2022. Chap. 5, pp. 59–75. ISBN: 978-3-031-07297-0. DOI: 10.1007/978-3-031-07297-0_5.

[9] Pardha Koyya, Young Lee, and Jeong Yang. “Feedback for programming assignments using software-metrics and reference code”. In: *International Scholarly Research Notices* (2013). DOI: 10.1155/2013/805963.

[10] Stephan Krusche, Mjellma Berisha, and Bernd Bruegge. “Teaching code review management using branch based workflows”. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. 2016, pp. 384–393. DOI: 10.1145/2889160.2889191.

[11] Zhiyu Li et al. “Automating code review activities by large-scale pretraining”. In: *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2022, pp. 1035–1047. DOI: 10.1145/3540250.3549081.

[12] Eduardo Oliveira, Shannon Rios, and Zhuoxuan Jiang. “AI-powered peer review process: An approach to enhance computer science students’ engagement with code review in industry-based subjects”. In: *ASCILITE Publications* (2023), pp. 184–194. DOI: 10.14742/apubs.2023.482.

[13] Ollama. *Ollama Open Source LLM Models*. Accessed: 2024.01.14. 2024. URL: https://ollama.ai/.

[14] Saikrishna Sripada, Y Raghu Reddy, and Ashish Sureka. “In support of peer code review and inspection in an undergraduate software engineering course”. In: *IEEE 28th conference on software engineering education and training*. IEEE. 2015, pp. 3–6. DOI: 10.1109/CSEET.2015.8.

[15] Deborah A Trytten. “A design for team peer code review”. In: *ACM SIGCSE Bulletin* 37.1 (2005), pp. 455–459. DOI: 10.1145/1047124.1047492.

[16] Wei Zheng, Yutong Bai, and Haoxuan Che. “A computer-assisted instructional method based on machine learning in software testing class”. In: *Computer Applications in Engineering Education* 26.5 (2018), pp. 1150–1158. DOI: 10.1002/cae.21962.

An Adaptive, Hint-Driven, Polymorphic CTF Framework for Educational Purposes*

David Pouliot[1] and Nate Balmain[2]
[1]Computer Science Department
Southern Oregon University
Ashland, OR 97520
pouliotd@sou.edu
[2]Computer Science Department
University of Oregon
Eugene, OR 97403
balmain@uoregon.edu

Abstract

Capture the flag (CTF) challenges are very popular among security professionals to demonstrate and practice their skills. However, there are numerous challenges when using most CTFs as an educational tool in the classroom. Most CTFs are too challenging for beginners, or their difficulty increases too steeply. The only recourse for these beginners is to find the complete solution online. Many CTFs use the same flag for each user, allowing users to share flags rather than solve the level themselves. Similarly, most CTFs deploy the exact same challenge to each user, allowing users to share scripts. Some CTF frameworks have attempted to remedy some of these shortcomings. To address these issues, we present Penetration-Platoon, An Adaptive, Hint-Driven, Polymorphic CTF Framework for Educational Purposes. While other systems have offered features like hint systems or polymorphic challenges to address these issues, our framework uniquely combines a dynamic hint system, polymorphic challenges, and the versatility to host any type of challenge in a unified package.

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Capture the Flag (CTF) challenges have gained widespread popularity among both computer security students and professionals. These challenges gamify the process of exploiting vulnerabilities, offering an enjoyable means of honing one's skills. However, while CTFs are commonly used as tests of knowledge and abilities, they often fall short as instructional tools for teaching these skills. As a result, many novice computer science students find it frustrating to get started with CTFs. When faced with challenges, their primary option is to resort to online walkthroughs for guidance, or to quit.

Most CTFs use the same flag for each user. While this is fine for many uses of CTFs, it is problematic when using a CTF as a graded activity for a course as users can share flags. For educational use, it is desirable for each user to have a distinct and unpredictable flag.

In many Capture the Flag (CTF) challenges, solving them is often facilitated by writing a short script. Similar to the concept of having distinct flags for each user, the ideal scenario would not allow a user to use another's script without making necessary adjustments to account for the unique conditions or parameters of their specific challenge instance.

The idea is to encourage participants to develop their own customized solutions, rather than simply copying and pasting pre-existing scripts. This approach not only reinforces the learning experience, but also aligns with the spirit of CTF challenges; which is to test and hone one's problem-solving skills in a hands-on and practical manner.

An ideal educational CTF framework should cater to beginners with a gentle learning curve and comprehensive hints, while remaining challenging for experienced users (they can choose not to use hints). It should feature unique flags per user, polymorphic challenges to prevent script sharing, as well as the ability to start a CTF without any complex setup. This approach promotes genuine learning, personalized experiences, and accessibility for all skill levels.

2 Background and Related Work

AutoCTF [4] is a specialized tool for creating exploitable program challenges in CTFs. It continuously generates new challenge instances with varying difficulty levels, allowing users to repeatedly practice specific exploit types. While limited to exploitable programs, AutoCTF enables users to hone their skills until they master each challenge category.

MetaCTF [3] is a specialized platform for reverse engineering challenges, featuring: Scaffolded learning with progressive difficulty levels, polymorphic and metamorphic code generation, unique flags and slightly modified challenges for each user. These features provide individualized learning experiences and prevent solution sharing. While currently limited to reverse engineering, MetaCTF's approach offers valuable training in code analysis skills.

Blinker [1] is a tool that will make numerous random changes to a binary executable. They implemented eight techniques for introducing variation into C/C++ binary challenges. This level of randomization is an excellent tool for enhancing the replayability of challenges.

PWN The Learning Curve [5] offers two key educational contributions: a set of CTF challenges with a gradual difficulty progression, preventing beginners from quitting due to overly steep learning curves, and a Jinja template system for adding randomness to challenges. This system allows easy generation of slightly different versions for each user or iteration, enhancing the learning experience and discouraging solution sharing.

Pwn Lessons Made Easy With Docker [6] designed a set of CTF challenges that aligns with the NSA CAE Cyber Operations Knowledge Units for vulnerability research. They utilized Docker containers for each challenge so they could select specific operating systems as well as specific version of that operating system. The challenges they created are the same for each user and they noticed suspicious behavior that led them to believe that students were sharing scripts to solve challenges. This indicates a real need for each user's challenge to have a slight variation.

3 Goals and Approach

Because of the shortcomings of using most CTFs for educational purposes, this project had several goals for an educational CTF framework:

- Develop a hint/help system tailored for CTFs, aimed at aiding users from beginners to experts, including a multi-level hint system complete with customization and tracking.

- Establish a flexible and configurable framework for deploying CTFs to cloud platforms, integrated with the hint system. This system supports multiple CTF Formats, accommodates different deployment scenarios, requires minimal setup and software for the user, and ensures each user receives a unique and random flag for each level. In addition, this framework also tracks the hint usage, simplifies the creation and deployment of CTF levels, and is optimized with cost savings in mind.

- Develop a method for introducing randomness into every CTF level. Ideally this would let the overall technique used to solve the CTF be the same for each user, but would vary on some of the implementation details. This randomness should enable each participant's challenge to have enough variation so that script sharing without modifying the scripts will not solve the challenge. In addition, if a user starts, stops and then restarts the challenge, they should receive the same challenge (though some levels might want to employ a truely random setup for replayablity).

- Develop CTF levels utilizing the hint system and level randomness. Deploy levels from within the framework.

4 Deployment

The foundation of our cloud-based system lies in its deployment. By utilizing a cloud platform such as Google Cloud, we eliminate many barriers to entry for beginners, requiring only a web browser and a terminal. This ensures that deployment is a seamless experience for users starting with CTF challenges. This section introduces the main web application, detailing the program architecture, and scoring system.

Main Web Application Our application is a web-based platform hosted on the Google Cloud infrastructure. Users interact with a web application to access and view information about Capture The Flag (CTF) challenges hosted on the platform. Our architecture aims to provide a scalable, secure, and cost-effective solution for hosting and managing CTF challenges by encompassing the following components: Cloud Run hosts the primary web application, Cloud Datastore–a fully managed NoSQL document database–is leveraged as the back-end database, VMs that host the CTF levels are provisioned from Compute Engine, Artifact Registry is used for storing and managing the CTF Level containers, Cloud Functions and Pub/Sub are utilized for resource management.

Flag Generation To facilitate unique flags, the web server generates a random seed when a student starts a level. In addition, this seed is stored in the database so that if a user stops the level and restarts it later, the same flag is re-generated. We chose this approach to alleviate user frustration, as it would be irritating to work on a multi-flag CTF level only to lose progress when the user wants to take a break.

The default flag generation method uses the hashing algorithm SHA3. We chose SHA3 due to its resilience against recovering its input parameters. The input to the hash function is the random seed concatenated with the username and the CTF level name. By concatenating multiple parameters, we avoid duplicate flags from occurring, even in the extremely rare case a random seed is generated twice. This ensures that all generated flags are unique and not predictable.

Hint System In order to satisfy the goals of the framework hint system, we decided to implement the hints on the web application rather than in the CTF containers. If the hint system was in the container, then the container would have to communicate with the web application in order to track how many hints a user obtained for a specific level. Since many of these containers are vulnerable by design, we did not want the hints stored on the containers at all. For security purposes we also did not want the potentially vulnerable containers to have any communication with the web application.

Thus, the user simply requests a hint for a level on the web application. The web application keeps track of how many hints each user requests for every level. These hints tie into the scoring system for the purposes of grading or scoring.

Our solution is not the first CTF solution that offers hints to its users. For example, Pico-CTF [7] also offers hints to their users to aid in the solution process. The area that makes us most unique, however, is our use polymorphic solutions to each CTF level combined with a highly configurable hint system and the ability to host any type of CTF.

Scoring System To determine the score a user receives after successfully finding the flags on a level, we utilize two primary factors: the level's base score (most influenced by difficulty) and the number of hints used (influenced by each hint's weight which is configurable by the instructor). By tying the base score to the difficulty, we are able to scale the upper bound of the user's score for more challenging levels. This makes more challenging levels more rewarding for the student, and reflects the user's progress in solving it. The number of hints used by the student is subtracted from the base score, with each hint weighted differently based on how much it reveals about the solution. This incentivizes students to use as few hints as possible to maximize their score on the level.

Flag Checking The main issue with using dynamically generated flags is the difficulty of checking if they are correct. In order to keep these flags secure, we do not store any flags that are generated. We need only store the seed used to create the flags in order to ensure that the flags are correct.

When the user submits a flag for checking, the system pulls the random seed for the student's current level from the database. It then utilizes the seed with same flag generation procedure to re-create the same random flag. In instances requiring multiple flags, we simply use a different seed that is generated from the original and run the flag generation algorithm once per each seed.

Start CTF After a student logs into the website, they can view all public challenges and start any CTF. Upon starting a level, the server-side code will start the VM and container, obtain its IP address, then will provide instructions for connecting to the container. In addition, this page allows the user to acquire hints and submit the flag.

Terminate Levels As the vulnerable CTFs are hosted on Google Cloud, leaving containers running for too long would only increase the cost to the instructor. In order keep costs down we supply three mechanisms for which levels can be terminated.

The first, and most often used, method to terminate a level is done by the student. The student may choose to stop the level which simply deletes the level from Compute Engine and saves their progress for later, due to the unique seed being stored in the database. In addition, the level is also automatically deleted when the student finishes solving it.

The second mechanism is performed automatically behind the scenes using Google Pub/Sub. At a configurable interval, Pub/Sub triggers a Cloud Function that checks for VMs that have been running for too long. When triggered, it checks a configurable metadata variable in each VM against how long the VM has been running. If the level has been running too long, it will be deleted.

A manual termination option is provided as a final safeguard, allowing instructors to delete any level at their discretion, complementing the primary cost-saving methods for VM uptime.

5 CTF Challenges

We chose to host all of our CTF challenges inside Docker containers, due to the portability and flexibility offered by containerized environments. Since a container can run nearly any application, our CTFs are not limited to a specific category. This makes them easy to deploy and also keeps all challenges isolated from each other and the main web application. This is very important because the environments used in many of the challenges requires turning off many of the security protections that modern systems provide.

Container Templates If our system lacked container templates, it would be little more than a web app to provide hints and start containers. To make creating the CTF challenges easier, we also created a few templates to use as starting points in creating levels. Currently we have two distinct templates, one for terminal access and the other for web/http access. In the future we plan to add a remote desktop template as well.

The Web template is a simple Flask application. To create a CTF with a web vulnerability to exploit, it is a simple matter to add a route and template to the Flask application with the specific vulnerability. For example, we created a website with a SQL injection vulnerability. The flag is added to a database when the container image starts, and the only way to retrieve the flag is to execute a SQL injection to retrieve it.

The terminal-based template includes a base Dockerfile and a bash script. The script has access to the flag (passed into the container at runtime), and it is up to the level designer to decide what to do with the flag.

For example, with one of our binary exploitation/buffer overflow challenges, we change the owner of the flag to a user that also owns the program with the buffer overflow vulnerability. Once the challenger compromises this executable, they can now view the flag because it has the same owner as the executable.

In another example, we hide the flag inside a random file in a file system with numerous folders and files. The idea of this is to teach the challengers how to use grep.

These terminal-based levels can be nearly any type of CTF. We have created several binary exploitation levels, cryptography levels, forensics levels and some basic levels on how to use a shell.

Container Parameters Container parameters are specified by the instructor when they create a level. We supply arguments in two methods, each used for its own purpose: as a program input argument passed to the container upon creation via stdin, and as metadata. Argument parameters are used by the container for the purpose of flag generation, level variation, for use in the level, or for some other developer-specified purpose. These arguments are protected, as they contain necessary information regarding the level's unique seed.

Metadata parameters are used by the virtual machine that hosts the vulnerable container. These provide instructions to Google Compute Engine for which hardware to use, which container should be loaded into the virtual machine, and how long to keep the VM running. These parameters do not need to be kept secret, as the only information they contain is related to general VM and container information.

Securing Level Containers Since the containers deployed for each level contain vulnerabilities, it is necessary to secure them in order to prevent the challenger from escaping the container. We use a variety of measures to secure the containers as much as possible such as removing extra commands, using additional firewall rules, limiting the lifetime of the container, and minimizing access to other resources with the use of Google Cloud Service Accounts.

We use part of the entrypoint script that runs when the container starts to remove unnecessary commands from the system that could be potentially abused for nefarious purposes, such as scp, apt, and dpkg. By removing these commands from the system, they become unavailable for the challenger to use. This technique of removing commands can be used for either terminal or web levels. However, the exact programs removed can vary depending on the nature of the level. For example, allowing the cat command is likely necessary for a terminal level and not for a web level. It is important to sufficiently restrict access, or even outright remove, as many commands as possible in order to sufficiently secure the container as much as possible.

Firewall rules were setup such that they allow normal access to the container depending on what type of level the container is used for. With web vulnerability levels, we restrict all incoming and outgoing TCP connections to only use ports 80 and 443, thus allowing the user to access the website and restricting potential communication with other servers.

For terminal-based levels, we disable SSH's default port 22, only allowing incoming and outgoing SSH sessions to be held on a separate port. Further, we also restrict all other TCP and UDP ports in order to further minimize the chances of communication with other outside services. This combination disallows students from using SSH to connect to the host VM and only allows access to the container.

In addition, we also create the Virtual Machines that run the containers in Google Compute Engine with a custom service account that was assigned a role with minimal permissions. This Google Service Account only has permissions to download Artifacts (containers), and disk read access for the host VM. This way if a user manages to compromise the VM, they would have very limited access to the cloud resources.

Adding Randomness to CTF Challenges While the framework creates unique flags per user per level to stop flag sharing, it by itself does not prevent the sharing of scripts to solve the level. However it does facilitate this.

To prevent the user from simply using another user's script to solve a level, the CTF level needs some randomness in the design. But we also want the user to be able to start a level, stop it, and then come back to the exact same level to continue their work.

The seeds and flags that are already generated can be used by the level designers to accomplish this. Here is one example from the Seed [2] project. In this lab the instructor can change a constant (BUF_SIZE) to make the attack slightly different from a previous version of the course:

```
/* Changing BUF_SIZE changes the layout of the stack. Change
*  this value, so students cannot use past solutions.
*  Suggested value: between 0 and 400 */
#ifndef BUF_SIZE
#define BUF_SIZE 85
#endif

int bof(char *str){
  char buffer[BUF_SIZE];
  /* The following statement has a buffer overflow problem */
  strcpy(buffer, str);
}
```

The startup script in the container could take the flag, and use it as a generator to pseudo-randomly select a value between 100 and 400, then update the BUF_SIZE value in the program. Thus the general technique to solve this level (buffer overflow) would be the same for each student, but each student would get a slightly different version of program to exploit. In this case, the location and size of some of the stack frames would vary for each student.

A similar mechanism can be used to change aspects of a web security challenge as well. For example we could use the seed to dynamically change the ids of HTML tags, specific URLs, the passwords of specific users, etc.

6 Results

To test this framework, several web-based levels were created that are similar to levels you might find in the Portswigger [8] web site.

Many of the levels on the Portswigger web site have been used in a web security course taught by the author. Portswigger is fantastic for learning about web vulnerabilities. However, using it for a classroom is problematic. There are many complete walkthroughs available online. Further, there is no easy way for a student to show that they have completed a Portswigger level. Taking a screenshot and producing a lab report was the technique for this course.

In the Fall of 2023, we substituted numerous web-based CTFs using this framework in place of the previously used PortSwigger CTFs. The result was that students were able to get help, rather than a complete solution, when they were stuck. Additionally, showing that a student completed a level, along with the number of hints they used, was automated, saving both the student and the instructor the time of creating and grading lab reports.

7 Conclusion

Learning cybersecurity does not have to be an arduous and tedious task. By leveraging engaging activities such as CTFs, the learning process becomes more fun and engaging for students. The highly configurable hint/help system of Penetration Platoon guides both novice and more experienced students. This approach reinforces learning better than looking up a solution online, or copying and pasting some code for an exploit. Using dynamic seeds keeps every level's solution unique, thereby eliminating flag and script sharing by students. Through the use of containers, we can host a CTF of any variety while keeping the vulnerabilities isolated. While many other previous works offered similar features such as a hint system or polymorphic challenges, our system combines a highly dynamic hint system, polymorphic challenges and the versatility to host any type of challenge in a unified package.

References

[1] *Blinker: automatic generation of computer security exercises.* `https://gs509.user.srcf.net/blinker/`. Accessed: 2024-05-01.

[2] Wenliang Du. "SEED: Hands-On Lab Exercises for Computer Security Education". In: *IEEE Security & Privacy* 9.5 (2011), pp. 70–73. DOI: `10.1109/MSP.2011.139`. URL: `https://seedsecuritylabs.org/`.

[3] Wu-chang Feng. "A Scaffolded, Metamorphic CTF for Reverse Engineering". In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. D.C.: USENIX Association, Aug. 2015. URL: `https://www.usenix.org/conference/3gse15/summit-program/presentation/feng`.

[4] Patrick Hulin et al. "AutoCTF: Creating Diverse Pwnables via Automated Bug Injection". In: *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. Vancouver, BC: USENIX Association, Aug. 2017. URL: `https://www.usenix.org/conference/woot17/workshop-program/presentation/hulin`.

[5] Connor Nelson and Yan Shoshitaishvili. "PWN The Learning Curve: Education-First CTF Challenges". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024. <conf-loc>, <city>Portland</city>, <state>OR</state>, <country>USA</country>, </conf-loc>: Association for Computing Machinery, 2024, pp. 937–943. ISBN: 9798400704239. DOI: `10.1145/3626252.3630912`. URL: `https://doi.org/10.1145/3626252.3630912`.

[6] TJ OConnor et al. “PWN Lessons Made Easy with Docker: Toward an Undergraduate Vulnerability Research Cybersecurity Class”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024. <conf-loc>, <city>Portland</city>, <state>OR</state>, <country>USA</country>, </conf-loc>: Association for Computing Machinery, 2024, pp. 986–992. ISBN: 9798400704239. DOI: `10.1145/3626252.3630911`. URL: `https://doi.org/10.1145/3626252.3630911`.

[7] *PicoCTF*. `https://picoctf.org/`. Accessed: 2024-05-01.

[8] *PortSwigger*. `https://portswigger.net/`. Accessed: 2024-05-01.

An Evaluation on the Impact of Large Language Models on Computer Science Curricula*

Junghwan Rhee, Aakankshya Shrestha, Gang Qian,
Fei Zuo, Jicheng Fu, Myungah Park, Xianshan Qu,
Goutam Mylavarapu, Hong Sung
Department of Computer Science
University of Central Oklahoma, Edmond, OK 73034
{jrhee2,ashrestha51,gqian,fzuo,jfu,mpark5,xqu1,smylavarapu,hsung}@uco.edu

Abstract

Since their introduction, large language model (LLM) services have been widely used in our society, including the computer science education area. While this technology provides various types of intelligent assistance to users, its capabilities and impact on computer science education regarding students' learning need further study. In this paper, we present our manual assessment of LLM services' ability to solve questions in various course assignments and projects in our computer science curriculum. Based on the result of the study, we provide our observations of the extent of LLM services' impact on different computer science disciplines. Suggestions are summarized and offered to computer science instructors on the possible strategies for dealing with LLMs in current and future computer science curriculum designs.

1 Introduction

Large language model (LLM) services such as ChatGPT, Gemini, and Microsoft Copilot [12, 14, 11] have been widely used and affecting our society

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

since their introduction. While these intelligent services benefit most areas, education is one of the areas that need attention because unregulated usage may cause lost opportunities for proper education and ethical issues. Especially one of key areas of LLMs is code generation as shown by specialized services like Github Copilot, OpenAI Codex, Amazon CodeWhisperer, Tabnine, and Deep Code (by Snyk). Therefore, the instructors in our department have been continuing discussions on what is our students' learning experience and what kinds of directions should our department take to get our curricula aligned with the new era of AI and stay competitive.

In this paper, we present our initial effort of a collaborative study of multiple instructors in the department (Computer Science) to evaluate how well modern LLM services solve samples of our course assignments and project questions. After that, our instructors evaluate the performance of LLM solutions and analyze the implications. The result became our preliminary study to provide inputs and set our department's directions to react to the LLM services in terms of our course improvement. This paper has several new contributions.

- We have manually evaluated LLM services' performance on their solutions to homework assignments and course projects for multiple computer science topics and disciplines by multiple faculty members who teach such courses in our institution.

- We present our observations from the study of which types of questions and courses are more or less affected by LLM services and strategies to go along with LLM services based on evaluation and comments by instructors.

2 Design

In this section, we present the design of our study. We start with a list of courses whose lecturers offer the course for assignments for evaluation. These problems are fed into a common set of LLM services. Their outputs are sent back to each instructor who will evaluate the accuracy of the LLMs' solutions. Instructors will grade LLMs' results as a percentage whose value is between 0 (0%, completely wrong) and 1 (100%, entirely correct). The evaluation of LLMs' solutions is aggregated and used to extract common insights.

Multiple instructors of our department participated in this study, which covers topics such as data structures, theory of computing, computer organization, programming, and cybersecurity at multiple levels including elementary courses, advanced courses, and the specialized courses with certain themes (e.g., fundamentals, advanced, I, II) from multiple instructors.

Course#	|Q|	LLM 1		LLM 2		LLM 3	
		AVG	STD	AVG	STD	AVG	STD
1	10	0.6	0.52	0.12	0.21	0.405	0.42
2	16	0.75	0.45	0.875	0.34	0.75	0.45
3	105	0.63	0.49	0.49	0.50	0.51	0.50
4	10	0.82	0.33	0.072	0.17	0.325	0.44
5	11	0.86	0.23	0.458	0.38	0.513	0.36
6	12	0.89	0.9	0.7	0.24	0.24	0.29
7	10	0.99	0	0.96	0.07	0.84	0.23

Table 1: Performance of Three LLM Services on Multiple Courses.

3 Evaluation

We evaluated the performance of three major LLM services (ChatGPT, Gemini, and Microsoft Copilot) on our curriculum mainly regarding assignments and projects. For all LLM services, we utilized the free versions. For ChatGPT, version GPT-4o is used for a limited number of questions then GPT-3.5 is used. Gemini 1.5 Pro and Microsoft Copilot (Free Version) based on ChatGPT 4 Turbo have been used at the evaluation period.

We collected and summarized the evaluation result in Table 1. For a course number (Course#) and the number of questions (|Q|), we measured the performance of three LLM services with average scores (AVG) and standard deviations (STD). We did not specify the course names here because the results will be greatly different depending on how instructors set up questions that may not properly represent relevant course characteristics. Instead, this table illustrates the diversity of LLM services' performance in different courses. For the course #3 with 105 samples, the performance of three LLM services is similar as shown in their average values while some courses such as course #6 show clear performance differences among LLM services. We note that due to the nature of LLM services' operations, there is randomness involved meaning even for the same question, the evaluation result may differ each time. Also, the performance of the result may vary over time as the LLM companies are continuously improving their services. This is the result based on the LLM versions as of the first half of 2024. To avoid any bias from our limited samples of evaluation, each instructor's choices of question styles, and the randomness of LLMs' results, we extracted observations mainly from instructors' inputs.

3.1 Higher Impact Question Types and Affected CS Disciplines

These question criteria are *easier to LLMs and more serious issues for CS instructors that need immediate attention.*

- **Questions that are relatively straightforward and/or directly concept-based.** Example: True or false: If L is a regular language and L' is a subset of L, then L' is guaranteed to be a regular language.
- **Programming questions involving basic to intermediate concepts.** Example: Write a C++ program using dynamic arrays, finding the `second highest` and the `second lowest` without sorting the list and not using macros.
- **Programming questions with requirements (i.e., prompts).** Example: Generate a search page based on the Flask platform.
- **Survey or analysis of knowledge retrievable from public resources (e.g., the Internet).** Example: Complete a survey of the details of Windows Management Instrumentation regarding what WMI is, where it is applicable, the usage APIs, and any unique characteristics.

Observation #1

Question Types: LLMs provide competitive solutions for concept-based questions, general programming assignments, and knowledge retrieval, comprehension, and summary from textbooks or public sources.

CS Disciplines: Most programming courses as well as courses using assignment questions involving surveys of information need attention due to LLM services' strong capability.

3.2 Lower Impact Question Types and Related CS Disciplines

These question criteria are *the harder types to LLMs giving instructors more time to react at this moment.* However, we believe this is only temporary. The current development and future LLM services are likely to surpass current limitations.

- **Logical reasoning questions.** Example: Using the CRC polynomial 1011 and the information word 01001101, (a) compute the CRC code word for the information word; i.e., the bit pattern that should be sent with added redundant bits. (b) When the CRC code word from (a) is received at the receiver, show that the received code word does not contain any error.

- **Writing regular expressions**. Example: Give a regular expression r such that it represents the set of strings over the alphabet $\{a, b\}$ with an even number of a's and any number of b's. The regular expression should include only concatenation, union, and Kleene Star. Use $+$ as the union operator.

- **Questions that require an understanding of a particular concept/property and then use logical reasoning.** Example: What can we conclude from the following statement? Possible answers include the follows: L is half-solvable, L is solvable, L is regular, Nothing, L is not regular, L is not solvable, and L is not half-solvable. Give the most precise answer possible. $L_1 \cup L_2 = L$ where L_1 is in LNFA and L_2 is regular.

- **Questions requiring code comprehension and subsequent logical analysis.** Example: What is the output of the following code snippet if the user provides `‘‘NEW YORK’’` as the input? `...code example...` This question requires understanding the code first and then deducing the correct answer based on the understanding.

- **Finding logical or general errors.** Example 1: When a function of an incorrect but similar class is used with a different class object, most LLMs fail. Example 2: If there is a code segment doing something that does not match with the condition in the conditional statement, most of LLMs have difficulties finding the error when running with multiple such test cases.

- **Questions about UML diagrams.:** Example: As of now, LLMs couldn't generate UML class, sequence, component, deployment, and use-case diagrams. In the future, even if LLMs can, due to the hallucination issue, they may not present the correct interpretations for software requirements,

- **Programming questions with advanced concepts.** Example: When more advanced concepts (such as inheritance, and polymorphism) are required in the problem statement, LLMs are producing the code that sometimes does not meet all the requirements.

Observation #2

Question Types: LLMs show weakness at logic-based questions such as regular expressions and logical reasoning, deep reasoning to find errors, code comprehension with logical analysis, programming with advanced concepts, and UML diagram generation at this moment.

CS Disciplines: Courses with logic theory such as the theory of computing, low division computer organization covering boolean logic, software engineering courses with UML, and programming courses with advanced concepts seem to have more time until LLMs catch up.

4 Multiple Directions for CS Courses

Based on our study, we have determined several suggestions regarding the strategies how to make our curriculum go along with LLM services.

4.1 Approaches to Promote LLM Usages

This approach is to promote LLMs and actively incorporate them into current courses to actively utilize their benefits.

- **Learn to be competent with high-level abstractions as a software architect:** As code generation is one of the strengths of LLMs, if students could provide the correct prompts (requirements), LLMs could generate correct (or almost correct) code. Therefore, we can help students raise the level of abstractions from coding to higher levels such as requirements engineering, software design, and project management.
- **Build LLM-based tools for guiding students to write better code:** Another approach is to create assistive tools that can give feedback to students, where the tool acts as a mentor for students. Woodrow et al. [15] developed a ChatGPT-based tool that would provide real-time “style feedback” to help students write better code. This real-time feedback was found to be five times more effective than delayed feedback by a human mentor. They encourage human monitoring and input with this approach since LLMs are known to hallucinate and provide wrong answers at times.
- **Utilize LLMs as an assistant and learn how to evaluate code, engineer prompts, and manage LLMs’ work:** An approach to utilizing LLM is to treat it like an assistant team member. An example

is to let students use LLM services to generate an intermediate output (e.g., code) as they take the roles of a reviewer, a mentor by engineering prompts, and a manager working iteratively for multiple rounds and generate the final outcome.

Direction #1

A student is encouraged or required to use LLMs as a software architect, an LLM evaluator, an LLM mentor, and a project manager.

4.2 Approaches to Cooperate with LLMs

This approach gives permission to use LLMs without promoting it or preventing it.

- **Use LLMs like a search engine or a calculator:** As students use a calculator or a search engine nowadays, we can give them permission to use LLM services to get assistance to complete assignments. However, the work should be students' own work. Students should pay effort to avoid plagiarism rewriting the result with their own language.
- **Discussion on ethics and capabilities of AI tools:** As the use of LLM services grows, the discussion on proper use and ethical concerns should increase in parallel. As educators regardless of our stance on using AIs in education, we must still understand new AI technology and their potential impacts [9]. Involving LLMs in academia is still new and evolving and there hasn't been any defined rule set to follow [1]. Students and faculty using LLM services need to be aware of the ethical concerns surrounding the use of AI.

Direction #2

Students are reminded to be responsible users doing their own academic work while they may use LLM services.

4.3 Approaches to Make Courses Resistant to LLMs

Another approach is to make the current course material valid and less vulnerable as long as possible with minimal impact from LLMs. Based on the study, we found current LLMs may not be strong enough to solve certain types of questions. If we transform existing assignment questions into such types, we can make them more resistant or confusing to LLMs.

Several experiments with LLMs have shown much more effective performance of LLMs after modifying the prompts. While we make the questions harder for an LLM to decode, students can change the questions so that the LLMs can understand and generate solutions for them. A recent 2023 study by Denny et al.[4] found that for CS 1-level questions, Copilot could solve an additional 31% of the problems after the prompt had been modified.

Direction #3

We can make courses more resistant to LLMs. However, students can reverse the effort with problem modification and prompt engineering.

4.4 Approaches to Discourage or Prevent LLM Usages

This approach forbids the usage of LLM services. Students will be notified to refrain from using LLMs for their assignments. Otherwise, any LLM usage will be a violation of course conduct. A related technique is to fingerprint/detect any LLM result and penalize students using the evidence. For instance, LLMs tend to draw UML diagrams in a textual format. Therefore, the textual format for UML diagrams is a strong indicator that students relied on LLMs to generate the UML diagrams.

However, with the fast growth of LLM services, this approach is likely to be temporary. As search engines, the Internet, or a calculator have become so common in our lives, it will be hard to prevent LLMs in the long term.

Direction #4

We may be able to discourage, penalize, detect and/or prevent LLM users for the moment. However, it will be hard in the long term.

5 Related Work

LLM's impact on Software Engineering. The evolution of LLMs like chatGPT and Copilot will change the way software engineers work, which in turn should reflect a change in software engineering education [8]. Researchers expect to see the software engineering field moving towards reading and evaluating AI-generated code rather than manually writing code [1][3]. Software engineers are expected to focus more on understanding the software goals and thus maintaining and debugging AI-generated code [3].

As the career itself is facing a change, the academic curriculum should accordingly change to adapt to these changes. The field requires dedicated time and research for incorporating LLM tools into software engineering programs for two main reasons: first is the lack of reliability of LLM models as they are prone to hallucination and second is that LLM models cannot deal with safety, privacy and physical reality [8].

The rapid development of AI tools is creating a new era where AI will impact our everyday lives [5]. A survey from 2023 by Tyna et al. [7] suggests that approximately 19% of jobs have at least 50% of their tasks exposed to LLMs.

Academia's usage of LLMs. ChatGPT alone has introduced multiple problems for educators as they can never be sure if students have used ChatGPT to solve the assignments [3]. However, in the past couple of years, many academic institutions around the world have made efforts to make use of LLMs in their curriculum. Denny et al. [4] explored the performance of copilot against publicly available coding questions and found that out of the total 166 questions, 79 of them were solved verbatim, 53 of them were solved after modifying the prompt and 34 remained unsolved [4]. Likewise, prompt engineering has also yielded favorable results in Data Science subjects. Shen et al. [13] found that for intermediate and advanced-level courses, ChatGPT increased its correctness by 45.65% and 63.98% respectively after modifying the prompts.

While others have leveraged tools using LLMs to help mentor students, Lan et al. [9] presented a model for using ChatGPT to create an AI agent using a series of complex prompts that helps students learn. Liu et al. experimented with creating a Teaching Assistant using GPT 3.5 from OpenAI [10]. They found the virtual TAs to be more engaging, clear, and detailed than human TAs while having similar levels of accuracy, though the answers of the virtual TAs require cross-checking for optimal and correct performance.

A literature review by Combaz et al. [2] suggests that while using AI tools as an "assistive tool" is encouraged, a manual review of AI-generated content is important. However, overuse of such tools can also cause over-reliance and reduced critical thinking [2]. Likewise, Duarte et al. [6] maintain that while we can encourage assistance from AI models, peer review and authorship should be done by humans as using AI can incur risks like algorithmic bias and even authorship by AI.

6 Conclusion

In this paper, we present our study on LLM services' impact on student learning experience. We used three major LLM services to solve our curricula's assignment and project questions and the results were manually evaluated by the course instructors qualitatively. We found out that current LLMs are very competitive at generating code, answering concept-based questions, and the retrieval of public information. On the other hand, solving logic-based questions or finding errors is challenging at this moment, but this could be changed anytime soon due to the fast evolution of LLM services.

We also discussed multiple directions to promote LLMs, cooperate with LLMs, be resistant to their usage, or prevent their usage completely. However, due to the strong growth of LLMs' capability, eventually, we will need to adopt LLMs as an essential component of curricula.

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 SIGCSE 2023.* <conf-loc>, <city>Toronto ON</city>, <country>Canada</country>, </conf-loc>, 2023. ISBN: 9781450394314. DOI: `10.1145/3545945.3569759`. URL: `https://doi.org/10.1145/3545945.3569759`.

[2] Doga Cambaz and Xiaoling Zhang. "Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review". In: *Proceedings of the SIGCSE 2024.* 2024, pp. 172–178. ISBN: 9798400704239. DOI: `10.1145/3626252.3630958`. URL: `https://doi.org/10.1145/3626252.3630958`.

[3] Marian Daun and Jennifer Brings. "How ChatGPT Will Change Software Engineering Education". In: *Proceedings of the ITiCSE 2023.* New York, NY, USA, 2023, pp. 110–116. ISBN: 9798400701382. DOI: `10.1145/3587102.3588815`. URL: `https://doi.org/10.1145/3587102.3588815`.

[4] Paul Denny, Viraj Kumar, and Nasser Giacaman. "Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language". In: *Proceedings of the SIGCSE 2023.* Toronto ON, Canada, 2023. ISBN: 9781450394314. DOI: `10.1145/3545945.3569823`. URL: `https://doi.org/10.1145/3545945.3569823`.

[5] Paul Denny et al. "Computing Education in the Era of Generative AI". In: *Commun. ACM* 67.2 (Jan. 2024), pp. 56–67. ISSN: 0001-0782. DOI: `10.1145/3624720`. URL: `https://doi.org/10.1145/3624720`.

[6] C. C. Duarte. "Authorship and Peer Review in the Era of Artificial Intelligence". In: *Computer* 56.12 (Dec. 2023), pp. 32–41. ISSN: 1558-0814. DOI: `10.1109/MC.2023.3311729`.

[7] Tyna Eloundou et al. *GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models*. 2023. arXiv: 2303.10130 [econ.GN].

[8] Vassilka D. Kirova et al. "Software Engineering Education Must Adapt and Evolve for an LLM Environment". In: *Proceedings of the SIGCSE 2024*. 2024, pp. 666–672. ISBN: 9798400704239. DOI: 10.1145/3626252.3630927. URL: https://doi.org/10.1145/3626252.3630927.

[9] Yu-Ju Lan and Nian-Shing Chen. "Teachers' agency in the era of LLM and generative AI: Designing pedagogical AI agents". In: *Educational Technology & Society* 27.1 (2024), pp. I–XVIII. ISSN: 11763647, 14364522. URL: https://www.jstor.org/stable/48754837 (visited on 06/19/2024).

[10] Mengqi Liu and Faten M'Hiri. "Beyond Traditional Teaching: Large Language Models as Simulated Teaching Assistants in Computer Science". In: *Proceedings of the SIGCSE 2024*. 2024, pp. 743–749.

[11] *Microsoft Copilot*. Microsoft Research. 2021. URL: https://www.microsoft.com/en-us/research/project/academic/articles/new-feature-cite/.

[12] OpenAI. *ChatGPT*. https://openai.com/research/chatgpt. 2020.

[13] Yiyin Shen et al. "Implications of ChatGPT for Data Science Education". In: *Proceedings of the SIGCSE 2024*. 2024. ISBN: 9798400704239.

[14] Gemini Team, Jeffrey Dean, and Oriol Vinyals. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL].

[15] Juliette Woodrow, Ali Malik, and Chris Piech. "AI Teaches the Art of Elegant Coding: Timely, Fair, and Helpful Style Feedback in a Global Course". In: *Proceedings of the SIGCSE 2024*. 2024, pp. 1442–1448. ISBN: 9798400704239. DOI: 10.1145/3626252.3630773. URL: https://doi.org/10.1145/3626252.3630773.

Investigation of Computing Transfer Students Success *

Cheyenne Ty[1], Kay Vargas[1],
Yun Wan[4], Xiwei Wang[5], Palvi Aggarwal[3],
Shebuti Rayana[2] and Sherrene Bogle[1]
[1]Dept. of Computer Science
California State Polytechnic University Humboldt
Arcata, CA 95521
`{cjt101,kv111,sab30}@humboldt.edu`
[2]Dept. of Mathematics, Computer & Information Sciences
SUNY Old Westbury
Old Westbury, NY 11568
`rayanas@oldwestbury.edu`
[3]Dept. of Computer and Information Sciences
University of Texas At El Paso
El Paso, TX 79968
`paggarwal@utep.edu`
[4]Dept. of Computer & Information Sciences
University of Houston-Victoria
Victoria, TX 77901
`wany@uhv.edu`
[5]Dept. of Computer Science
Northeastern Illinois University
Chicago, IL 60625
`xwang9@neiu.edu`

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Abstract

Community colleges provide accessibility in the pursuit of higher education, especially for women, underrepresented minority (URM) students, and first-generation college students (FGCS). This is especially true for majors within the STEM field, such as computer science (CS). However, there is a disconnect between the rates of enrollment at community colleges and the prevalence of these underrepresented demographics in 4-year institutions and the CS industry. The literature suggests that social and institutional factors affect transferring CS majors. This study aims to provide an empirical analysis on the different factors affecting the academic performance of transferring CS students. The study found that although all measured demographics impacted post-transfer GPA, the largest disparities are attributed to race/ethnicity, especially among URM students. These results will help inform an AI-driven counseling system that caters towards transfer students in CS.

1 Research Problem

Community colleges (CCs) allow higher education to be more accessible to underrepresented students, and transfer systems allow these students to continue their education at 4-year institutions. In particular, women, underrepresented minority (URM), and first-generation college students (FGCS) have been the focus in the impact that the transfer pathway has. Jackson et al. (2013) notes that in 2008, around 49% of female bachelor's and master's degree recipients in STEM attended a CC, with similar percentages reported for multiple URM ethnicities [11]. Additionally, FGCS are more likely to start upper education at a CC in the hopes of transferring [7]. Considering the accessibility that CCs provide women, URM students, and FGCS, supporting them through their post-transfer academics may help to decrease the disparities in representation within the STEM workforce.

In particular, these disparities can be seen in the field of computer science (CS). Between 2017 and 2019, 25% of people working in the U.S. CS industry were women [9]. This report also notes that only 7% and 8% of CS workers in the U.S. were Black or Hispanic, compared to these groups making up 11% and 17% of workers across all fields. Intersections between underrepresented groups may also heighten disparities; less than 10% of CS Bachelor's degrees were handed to URM women in 2006 [8], while first-generation women in CS often display a greater lack of support and self-efficacy during their undergraduate years [3]. To foster a workforce representative of the overall population, it's important to focus on factors that may influence the academic success of these three underrepresented groups: women, underrepresented minorities, and first-generation students. This includes unique barriers that may exist for underrepresented transfer students, or the differences that are inherent between different transfer institutions. The current paper aims to provide a quantitative analysis of the impact that gender, URM, first-generation status, and transfer institution may have on the academic success of underrepresented transfer students in the CS field.

2 Review of Related Literature

2.1 Transfer Impact

Transfer students often face challenges that native (non-transfer) students in the institution do not. This can result in transfer students feeling less satisfied with their post-transfer institution, which is further amplified for students who are also both URM and FGCS [10]. One such challenge is the struggle to socially integrate into a post-transfer student body. Social circles among native students may already exist, making establishing social connections more challenging. Simultaneously, stigma within the native student body against transfer students' academic abilities may further reinforce feelings of social isolation and work to prevent integration [2]. These challenges can be further complicated by the intersectionality of transfer status and being underrepresented in their field, as is the case with women, URM students, and FGCS [2, 4, 16]. Transferring into an environment that lacks adequate support for social integration and academic goals for underrepresented students may hinder academic success in the post-transfer institution.

2.2 Institutional Differences

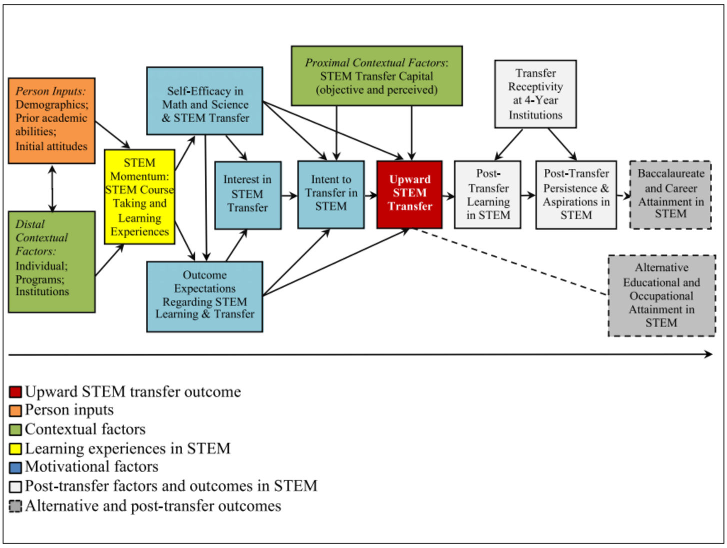
The different ways 4-year institutions treat transfer students may impact how well they integrate into the native student body. Institutions focusing on socialization among native, freshman students can have the unintended consequence of isolating transfer students [2, 4]. Opportunities for research and internships are also scarcer for transfer students, since faculty may not form as close of a relationship with transfer students compared to native students [18]. Conversely, providing unique transfer experiences or facilities catered towards underrepresented populations can help foster integration amongst the native population. Majka et al. (2021) details a boot-camp experience that helped develop communication and acceptance between transferring STEM students and native upperclassmen and faculty [13]. Spaces or facilities on campus that provide support for women, URM students, or FGCS can also help said students feel more integrated and reduce feelings of isolation in their new institution [14]. Hence, the programs and facilities that different 4-year institutions provide can have significant impact on the experience of underrepresented transfer students in both positive and negative regards.

2.3 Conceptual Framework

Wang's STEM Transfer Model [21] aims to identify the factors that influence upward STEM transfer into 4-year institutions. The model depicted in Figure 1, is also applicable to the persistence and performance of transfer students after successfully transferring. Wang's STEM Transfer Model was chosen instead of models like the Social Cognitive Theory (SCT) or Social Cognitive Career Theory (SCCT) since it provides a more specific framework for understanding the potential institutional, social, and personal factors in STEM transfer success. Its scope is also specific to CC transfer students, aligning with our research demographic. We focus on two aspects of the STEM Transfer Model: Person Inputs and Transfer Receptivity.

Person Inputs refers to characteristics such as demographics and previous academic success. Perceived stereotypes related to these characteristics can impact a student's self-efficacy within STEM, both before and after transferring. This can be seen in Cheryan et al. (2020), where perceived threats to one's identity may be enough to deter women from pursuing or continuing to pursue a CS degree [6].

Transfer Receptivity refers to the amount of support that an institution provides STEM students post-transfer. Institutional support can influence a transfer student in continuing their education and obtaining a Bachelor's degree, while a lack in institutional support can hinder or even halt this pursuit.



Person Inputs: Demographics; Prior academic abilities; Initial attitudes
Distal Contextual Factors: Individual; Programs; Institutions
STEM Momentum: STEM Course Taking and Learning Experiences
Self-Efficacy in Math and Science & STEM Transfer
Outcome Expectations Regarding STEM Learning & Transfer
Interest in STEM Transfer
Intent to Transfer in STEM
Proximal Contextual Factors: STEM Transfer Capital (objective and perceived)
Upward STEM Transfer
Transfer Receptivity at 4-Year Institutions
Post-Transfer Learning in STEM
Post-Transfer Persistence & Aspirations in STEM
Baccalaureate and Career Attainment in STEM
Alternative Educational and Occupational Attainment in STEM
🟥 Upward STEM transfer outcome
🟧 Person inputs
🟩 Contextual factors
🟨 Learning experiences in STEM
🟦 Motivational factors
□ Post-transfer factors and outcomes in STEM
▨ Alternative and post-transfer outcomes

Figure 1: The STEM Transfer Model, as seen in Wang 2016 [21]. The horizontal arrow throughout the entire diagram represents time.

3 Methodology

This study uses data derived from five different U.S. institutions: University of Houston-Victoria (Texas), Cal Poly Humboldt (California), Northeastern Illinois University (Illinois), University of Texas El Paso (Texas), and SUNY Old Westbury (New York). They are referred to as Institutions 1 through 5 within this paper, in the order of their appearance.

3.1 Integration of Framework

The data consists of CS students who have successfully transferred into a 4-year institution, aligning with the group of people the STEM Transfer Model is aimed at. The statistical analysis takes into account information such as gender, race/ethnicity, first-generation status, and earned transfer credits, reflecting Person Inputs prior to transferring. The institution a student transferred into is included as another variable in our models, highlighting potential institutional differences in Transfer Receptivity.

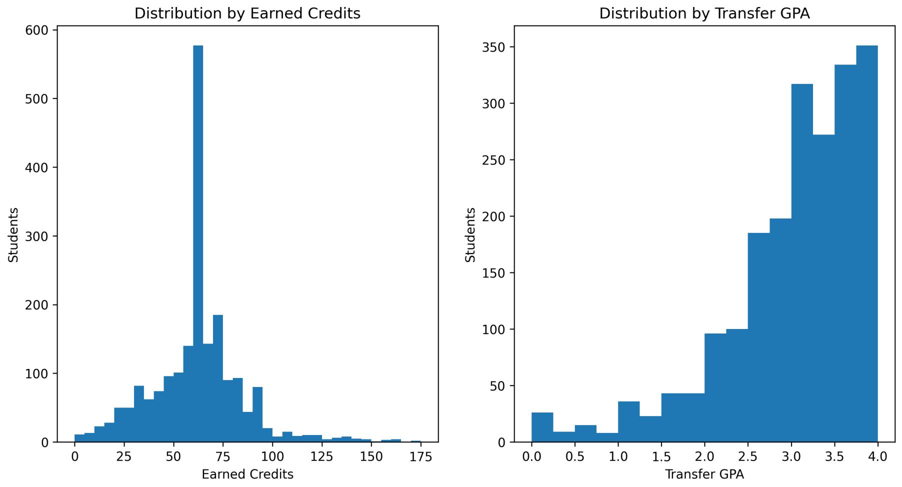


Figure 2: Student distribution based on earned transfer credits and transfer GPA. Five students with more than 175 earned credits are omitted in the first histogram, but are included in the final dataset.

3.2 Data Collection Method

The institutional data was obtained from the five institutions' respective registrar offices. The data was anonymized and filtered to only include CS and CS-related majors, with the most recent data for each student being kept. The comprehensive dataset was narrowed down to the following predictors: Transfer credits, gender, race/ethnicity, and post-transfer GPA. Only students who identified as either male or female were kept in the final dataset. Gender is not a binary label; however, the data had a small number of non-binary students ($N = 14$) and all of these students belonged to the same institution. Hence, to prevent bias we omitted them from this study. Additionally, post-transfer GPA refers to the GPA calculated using courses across all semesters and institutions.

Race/ethnicity was generalized into three categories: URM, non-URM, and Other. Non-URM consists of students who identified as only White or only Asian, since there is an overrepresentation of these ethnicities in CS [9]. URM consists of all other recorded races/ethnicities, which include African American, Hawaiian Pacific, Indigenous, and Hispanic/Latinx. Although the experiences of each URM group are unique in their own right, there is evidence suggesting that academic disparities exist that are consistent across races/ethnicities that fall under the URM category [1]. The URM category also includes undocumented students; although this is not a race or ethnicity, some institutions recorded it as such and the experiences that undocumented students face are very similar to the experiences that URM students face [14]. The Other category consists of students who either identified as being part of two or more races, of an unknown race/ethnicity, or students who put "Other" as their race/ethnicity. This grouping serves to represent the sample student population more accurately, accounting for the unique experiences of students who do not clearly fall under URM or non-URM. There also exists evidence that disparities URM students face may not be identical to the experience that students with unknown ethnicity have, as seen the large differences in attrition rates between URM and unknown ethnicity students in STEM [5]. The final dataset totals to 2056 students. Distributions of students by earned credits and transfer GPA can be seen in Figure 2, and a table of student demographics can be seen in Table 1.

Dem. \ Inst.	1	2	3	4	5	Total
Male	48	93	946	121	473	1681
Female	14	19	208	27	107	375
White	17	58	269	40	54	438
Asian American	12	9	227	32	13	293
African American	12	3	81	36	19	151
Hispanic/Latinx	7	22	328	33	469	859
Indigenous	0	1	2	2	1	6
Hawaiian Pacific	0	0	6	0	0	6
Undocumented	0	1	70	0	0	71
Two or More Races	10	12	22	1	8	53
Unknown	4	6	149	4	9	172
Other Ethnicity	0	0	0	0	7	7
Total	62	112	1154	148	580	2056

Table 1: Student demographics across all five institutions.

3.3 Data Analysis

Two sets of models were created using this data. For Set 1, a factorial ANOVA and a least-squares linear regression were performed. Post-transfer GPA was the response variable for both models, and predictor variables were race/ethnicity, gender, and institution. The linear regression also includes earned transfer credits as a predictor variable to represent past academic success, which has been shown to predict STEM persistence [8, 7]. Both models use all 2056 students from the final dataset.

Set 2 is similar to the models from the first set, but also includes first-generation status. Data used for this analysis only comes from Institutions 1 through 3, since first-generation status was only recorded for these three institutions. Since approximately one third ($N = 387$) of the data points report an unknown first-generation status, data imputation was performed using the k-nearest neighbors algorithm. A value of $k = 1$ was used to preserve the binary categories of first-generation and continuing-generation.This subset of the data includes 1169 students.

Factorial ANOVAs were conducted through the Pingouin Python package [20], and least-squares linear regressions were performed using the ISLP and statsmodels packages [12, 17]. The k-nearest neighbors algorithm was conducted using the scikit-learn package [15]. Significance was evaluated at the $p < 0.05$ level.

4 Results

4.1 Set 1

The ANOVA found that all main effects are significant on post-transfer GPA for the predictors of gender ($F(1, 2027) = 8.152, p = 0.0043$), institution ($F(4, 2027) = 205.937, p < 0.0005$), and race/ethnicity ($F(2, 2027) = 13.488, p < 0.0005$). The interaction between institution and race/ethnicity was also significant, $F(8, 2027) = 2.274, p = 0.0202$. The main effect of institution accounts for 28.25% of the variance ($\eta^2 = 0.2825$). The main effects of gender, race/ethnicity, and interaction between institution and race/ethnicity each account for less than 1% of the variance in GPA.

The conducted linear regression found that being a URM student significantly predicted post-transfer GPA ($\beta = -0.2211, p < 0.0005$). The amount of earned transfer credits also significantly predicted GPA ($\beta = 0.0035, p < 0.0005$). All other variables and interactions were deemed insignificant at the $p < 0.05$ level.

Earned transfer credits were significant in the linear regression as expected, since previous academic success may predict STEM persistence and achievement [8, 7]. Although race/ethnicity accounted for 1% of the variance in the ANOVA, its statistical significance alongside URM status being significant in the linear regression imply that race/ethnicity impacts post-transfer GPA to a degree. Furthermore, a URM student's GPA is predicted to be 0.22 points lower than a non-URM student's GPA, with all other variables constant. Institution having a high effect on post-transfer GPA in the ANOVA may be attributed to the uneven sample sizes for each institution, as seen in Table 1. As outlined in Figure 2, earned credits has inherent bias since most transfer students will have around two years of credits before transferring,

Variable	DF	F	p	η^2
Institution	2	9.921	<0.0005	0.0091
Ethnicity	2	382.430	<0.0005	0.3513
Gender	1	6.607	0.0103	0.0030
Institution * Ethnicity	4	7.030	<0.0005	0.0129
Ethnicity * Gender	2	69.858	<0.0005	0.0642
Institution * Ethnicity * Gender	4	8.079	<0.0005	0.0148
First Gen. * Ethnicity * Gender	2	11.398	<0.0005	0.0105
Institution * First Gen. * Eth. * Gender	4	3.354	0.0097	0.0062

Table 2: Variables with significant effects for Set 2's ANOVA, evaluated at the $p < 0.05$ level. The residual degrees of freedom was 1134.

especially among CC transfers. To better determine the effects of the post-transfer institution, more even sample sizes may be preferred. Additionally, pre-transfer GPA may provide a better measure of previous academic success, if it is available.

4.2 Set 2

With the inclusion of first-generation as a predictor, Table 2 shows all predictors that were deemed significant at the $p < 0.05$ level. Of note is that the main effect of ethnicity accounts for 35.13% of the variance ($\eta^2 = 0.3513$). Additionally, the interaction between race/ethnicity and gender yields a moderate effect size, accounting for 6.42% of the variance ($\eta^2 = 0.0642$). Two interactions involving first-generation status were significant, but the main effect was not ($F(1, 1134) = 0.104, p = 0.7468$).

The conducted linear regression found that being a URM student significantly predicted post-transfer GPA ($\beta = -0.1802, p = 0.005$). Earned transfer credits were also significant in predicting post-transfer GPA ($\beta = 0.0046, p < 0.0005$). The interactions that significantly predicted GPA were between URM ethnicity and first-generation status ($\beta = -0.1424, p = 0.025$) and between gender, first-generation status, and URM ethnicity ($\beta = -0.1789, p = 0.005$).

The results indicate that although first-generation status may not have a significant effect on post-transfer GPA by itself, its intersection with other underrepresented groups can amplify effects on post-transfer GPA. Race/ethnicity is of significance in this set of models as well, accounting for a high percentage of the variance in the ANOVA and URM status being significant in the linear regression. This consistency further indicates that race/ethnicity is of importance when considering the factors we have taken into account, though intersectionalities between underrepresented groups can still be worth noting.

5 Conclusion

This paper first reports on the effects that institution, gender, first-generation status, and race/ethnicity can have on transfer student success, and why certain groups are underrepresented in the CS academic field. Using themes from Wang's STEM Transfer Model, two sets of ANOVA and linear regression models were created to analyze the effects that these demographics can have on post-transfer GPA. The results suggest that although every demographic analyzed had some effect on a student's post-transfer GPA, URM status had the greatest effect. Intersectionality between being URM and another underrepresented group further amplifies gaps in GPA. Based on these results, more support towards URM students in particular may bridge the largest gaps in GPA for the transfer student population. Future analyses using pre-transfer GPA or additional data on students' pre-transfer academic success may yield clearer results on the effects of each demographic. Ultimately, these results will help design an AI-driven counseling system catered towards transfer students and tailored to their specific needs [19]. We expect that this system will allow CS transfer students to realize their full potential within higher education.

6 Acknowledgments

This material is based upon work supported by the National Science Foundation under CNS-2219623. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the National Science Foundation.

References

[1] David J. Asai. "Race matters". In: *Cell* 181.4 (2020), pp. 754–757.

[2] Peter Riley Bahr et al. "A review and critique of the literature on community college students' transition processes and outcomes in four-year institutions". In: *Higher Education: Handbook of Theory and Research: Volume 28* (2013), pp. 459–511.

[3] Jennifer M. Blaney and Jane G. Stout. "Examining the relationship between introductory computing course experiences, self-efficacy, and belonging among first-generation college women". In: *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education.* 2017, pp. 69–74.

[4] Jennifer M. Blaney et al. "Transfer Student Receptivity in Patriarchal STEM Contexts: Evidence of Gendered Transfer Student Stigma in Computer Science From a Mixed Methods Study". In: *Community College Review* (2024), p. 00915521231218233.

[5] Marguerite Bonous-Hammarth. "Pathways to success: Affirming opportunities for science, mathematics, and engineering majors". In: *Journal of Negro Education* (2000), pp. 92–111.

[6] Sapna Cheryan et al. “Double isolation: Identity expression threat predicts greater gender disparities in computer science”. In: *Self and Identity* 19.4 (2020), pp. 412–434.

[7] Sandra L. Dika and Mark M. D’Amico. “Early experiences and integration in the persistence of first-generation college students in STEM and non-STEM majors”. In: *Journal of Research in Science Teaching* 53.3 (2016), pp. 368–383.

[8] Lorelle Espinosa. “Pipelines and pathways: Women of color in undergraduate STEM majors and the college experiences that contribute to persistence”. In: *Harvard Educational Review* 81.2 (2011), pp. 209–241.

[9] Richard Fry, Brian Kennedy, and Cary Funk. “STEM jobs see uneven progress in increasing gender, racial and ethnic diversity”. In: *Pew Research Center* 1 (2021).

[10] Melissa Hawthorne and Adena Young. “First-Generation Transfer Students’ Perceptions: Implications for Retention and Success”. In: *Journal of College Orientation, Transition, and Retention* 17.2 (2010).

[11] Dimitra Lynette Jackson, Soko S. Starobin, and Frankie Santos Lanaan. “The Shared Experiences: Facilitating Successful Transfer of Women and Underrepresented Minorities in STEM Fields.” In: *New directions for higher education* 162 (2013), pp. 69–76.

[12] Gareth James et al. *An introduction to statistical learning: With applications in python.* Springer Nature, 2023.

[13] Elizabeth A. Majka, Merrilee F. Guenther, and Stacey L. Raimondi. “Science bootcamp goes virtual: a compressed, interdisciplinary online CURE promotes psychosocial gains in STEM transfer students”. In: *Journal of Microbiology & Biology Education* 22.1 (2021), pp. 10–1128.

[14] Mayra Nuñez Martinez. “Cultivating transfer receptivity for undocumented and DACAmented Latina/o/x students at 4-year institutions.” In: *Journal of Diversity in Higher Education* (2023).

[15] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[16] Marie-Elena Reyes. “Unique challenges for women of color in STEM transferring from community colleges to universities”. In: *Harvard Educational Review* 81.2 (2011), pp. 241–263.

[17] Skipper Seabold and Josef Perktold. “statsmodels: Econometric and statistical modeling with python”. In: *9th Python in Science Conference.* 2010.

[18] Bertin Solis and Richard P. Durán. “Latinx community college students’ transition to a 4-year public research-intensive university”. In: *Journal of Hispanic Higher Education* 21.1 (2022), pp. 49–66.

[19] Northeastern Illinois University. *AI-driven Counseling System for Transfer Students*. URL: `https://cs.neiu.edu/acosus/Home.html`.

[20] Raphael Vallat. “Pingouin: statistics in Python”. In: *The Journal of Open Source Software* 3.31 (Nov. 2018), p. 1026.

[21] Xueli Wang. “Upward transfer in STEM fields of study: A new conceptual framework and survey instrument for institutional research”. In: *New Directions for Institutional Research* 2016.170 (2016), pp. 49–60.

Résumé Revisions to Document Learning Outcomes in a Computational Biology Course*

Tammy VanDeGrift
Computer Science, Shiley School of Engineering
University of Portland
Portland, OR 97203
vandegri@up.edu

Abstract

Computer science courses, and higher education courses in general, have learning outcomes for skills, knowledge, and mindsets. These outcomes are usually publicized on course syllabi, websites, labs, projects, and other assignments. Faculty often assess learning outcomes on specific assignments, exam questions, and projects. This paper presents an assignment in which students document the skills and knowledge they developed in a course through several revisions of a résumé. The purpose of this assignment is not to assess learning outcomes directly, but to see how students characterized their experiences in the course. Even though students were not explicitly asked to include skills tied to course learning outcomes, all course learning outcomes appeared on at least one student's résumé. The learning outcomes that were more general (implementing/extending solutions, tracing/executing algorithms, teamwork/collaboration) appeared often on students' résumés. In addition to résumé revisions, the assignment required students to apply to at least one job, internship, or summer research experience. The résumés helped the instructor learn about students' backgrounds, goals, and interests and could advise them toward more specific internships and job opportunities throughout the semester. The assignment also showed the instructor what made it into the résumé, which were likely the course experiences that students valued the most.

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

College courses help students build knowledge, gain experiences, and explore interests. Computer science learning is often done through readings, videos, assignments, projects, labs, and in-class activities. While course learning outcomes are usually transparent, what students experience and value about courses may not be as visible to the students and to the instructor.

This paper presents a résumé assignment that benefits both the instructor and the students: the instructor sees what students value and students practice summarizing newly acquired skills and knowledge. The assignment was designed so students could document their learning journey; furthermore, the assignment had other unanticipated benefits. First, even though this course enrolled third and fourth year college students, a few students had not yet prepared a formal résumé. This assignment required them to build their first formal résumé. Second, the instructor provided feedback about potential résumé improvements. Third, the instructor got to know students early in the term since the first version of the résumé was due at the beginning of the second week of the semester. The résumés helped the instructor create project teams to include diverse skills and expertise among teammates. Fourth, the assignment may have helped students get internships and jobs, since one part of the assignment was to use the résumé to apply for at least one internship or job. Lastly, the instructor could see which skills and knowledge students valued the most based on what students added to their résumé. The résumé represents a condensed version of a person's experiences, so it naturally documents what the person deems as most valuable.

The remainder of this paper is organized into several sections. Section 2 discusses how résumés are used in other college courses, how résumés can tie to larger curricular goals, and how résumés are supported by educational theory. Section 3 describes the university, course, and the assignment. Next, Section 4 details the questions that were studied, showcases students' employment after the semester, and describes how the résumés documented learning outcomes. Section 5 concludes the paper with a summary and recommendations.

2 Related Work and Educational Theory

Résumés have been part of higher education, appearing in both research studies and as course deliverables. Petersheim and others studied computer science students' understanding of the résumé screening process [7]. Both CS students and recruiters screened résumés. Students generally moved more résumés to the next screening stage and students overestimated prior work experience as compared to recruiters. Berdanier and her colleagues studied engineering résumés for writing style in order to develop better pedagagogical content in technical writing courses [2]. Crowne and others share an educational model that includes résumés and career preparation in business courses in paper [5]. The educational literature includes several instances of using résumés in business, engineering [3], and chemistry courses [10]. Thummaloor and Mutyala designed a course to help CS students prepare a résumé and consider job opportunities [9]. This paper is complementary to these projects in that this work focuses on using résumés to document skills and experiences in a course..

Professional organizations recognize the importance of preparing students for professional practice. For example, ABET accredits computer science programs and one of the student outcomes is *Communicate effectively in a variety of professional contexts* [1]. This résumé assignment could be used as an assessment tool to evaluate if students can communicate effectively in a professional context. The ACM CS curriculum 2023 has a list of dispositions for CS graduates: adaptable, collaborative, meticulous, persistent, proactive, responsible, and self-directed [4]. The résumé assignment could help students practice the dispositions of being meticulous, proactive, and self-directed.

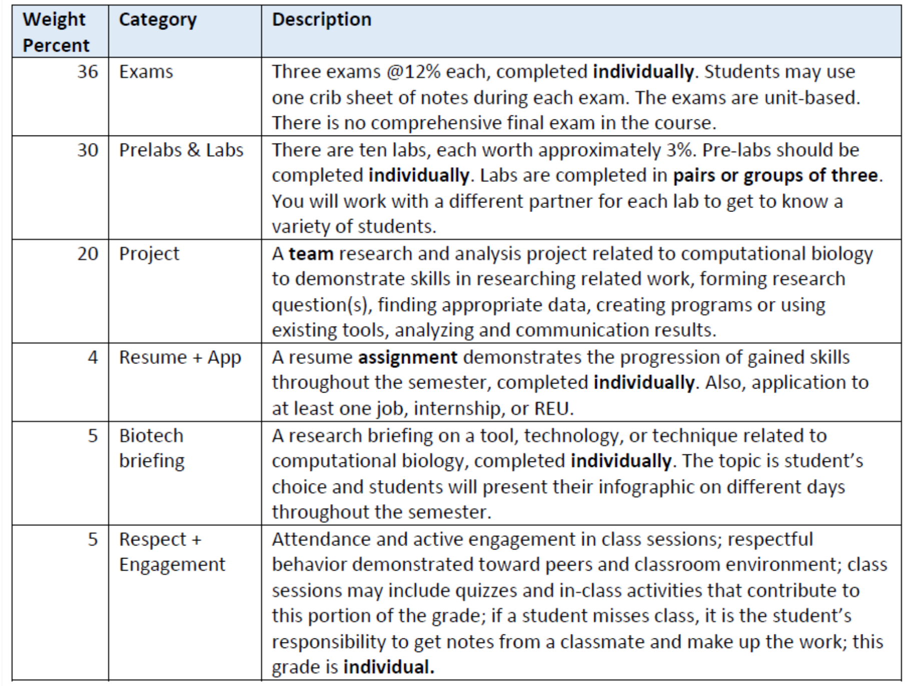
The learning theory of metacognition states that learners should reflect on how they learn, so they can better plan, monitor, and evaluate their own learning [8]. There are many examples of educational interventions that promote metacognition in computing students. For example, Mani and Mazunder asked students to rate their confidence of answers on exams to reflect on their own learning and understanding of the material [6]. The use of résumé revisions can provide a process to scaffold metacognition in students, so they can evaluate what was important in their learning.

3 Educational Context

The résumé assignment was part of a Computational Biology course at the University of Portland. The University is a small, private, regional university in the USA, offering majors and minors in Computer Science and Biology.

The Computational Biology (CB) course is a cross-disciplinary course, offered both as CS and BIO. The prerequisite structure supports multiple pathways into the course. Biology students must take an introduction to CS course and a cell/molecular biology course prior to CB. Computer science students must take Data Structures prior to CB. Therefore, the computing background can vary from one course to many courses. The CS students may have no biology experience. The instructor reminds students of the diverse backgrounds and to help by sharing their own expertise throughout the semester.

The full set of course assessments along with grade contributions is shown in Figure 1. The résumé and job application assignment, a small component of



Weight Percent	Category	Description
36	Exams	Three exams @12% each, completed **individually**. Students may use one crib sheet of notes during each exam. The exams are unit-based. There is no comprehensive final exam in the course.
30	Prelabs & Labs	There are ten labs, each worth approximately 3%. Pre-labs should be completed **individually**. Labs are completed in **pairs or groups of three**. You will work with a different partner for each lab to get to know a variety of students.
20	Project	A **team** research and analysis project related to computational biology to demonstrate skills in researching related work, forming research question(s), finding appropriate data, creating programs or using existing tools, analyzing and communication results.
4	Resume + App	A resume **assignment** demonstrates the progression of gained skills throughout the semester, completed **individually**. Also, application to at least one job, internship, or REU.
5	Biotech briefing	A research briefing on a tool, technology, or technique related to computational biology, completed **individually**. The topic is student's choice and students will present their infographic on different days throughout the semester.
5	Respect + Engagement	Attendance and active engagement in class sessions; respectful behavior demonstrated toward peers and classroom environment; class sessions may include quizzes and in-class activities that contribute to this portion of the grade; if a student misses class, it is the student's responsibility to get notes from a classmate and make up the work; this grade is **individual.**

Figure 1: Computational Biology course assessments.

the overall grade, was posted to the course management site at the beginning of the semester. Below is the text of the assignment:

> Background: Computational biology (also called Bioinformatics) is a field that combines systems thinking, systems integration, statistics, mathematical modeling, algorithms, software skills, and biological understanding. The course attracts students with a wide range of interests and skills. In order for the instructor to get to know you a little bit better, you will submit your résumé (and updates) throughout the semester. You will also apply to at least one job, internship, or REU that relates to computational biology in some way. Note: you do not need to move further in the job application process for this assignment.

Grading was based on completion and not quality. Students who had already committed to a summer internship or full-time job could submit an offer letter (with details omitted) instead of applying to a job/internship/REU. The deliverables, dates, and point values were as follows:

- Original Resume, due Jan 22, 2 points
- Comp Bio Resume v1, due Feb 21, 1 point
- Apply to job/internship/REU, due Feb 21, 2 points
- Comp Bio Resume v2, due Apr 5, 1 point
- Comp Bio Resume v3, due Apr 26, 2 points

4 Evaluation

The questions guiding this work are: 1) Did the résumé and job application assignment impact students' employment? and 2) How did résumé content reflect course learning outcomes? These questions are answered in the subsections below.

4.1 Q1: Students and Summer Employment

The course had 14 students enrolled, two of which were taking the course as auditors. Thus, a total of 12 students took the course for credit and are the participants of this study. Of the 12, eight were computer science majors, two were biology majors, and two were electrical engineering majors.

Eight of the 12 students completed all five components of the assignment. One student earned 50% (did not complete v1, v2, and v3), one student earned 62.5% (did not complete v2 and v3), one student earned 75% (did not complete v3), and one student earned 87.5% (did not complete v2). The completion rates may have been low for some students since this assignment was worth a total of 4% of the overall grade; the grade impact may have been worth the time saved. Three of the four students who did not submit at least one deliverable were graduating seniors, so they may not have valued updating a résumé with a full-time job already secured.

Table 1 shows that 10 of the 12 students had summer or full-time employment after the CB course. Note that four of the employers are in computational biology, indicated with + in the table. The application deliverable included the following companies: Leatherman Tool Group, HDR, Oregon Health Sciences University (three applications), FM Global, Stanford Bioinformatics, Pacific Northwest National Laboratory, and Think-Cell. Students 3 and 5 had internship offers and Student 6 had a full-time offer before Feb 21, when the assignment was due. Student 6 did not submit written proof of the offer, so did not earn the points for this part of the assignment. Table 1 has a star in column three if the student's application led to employment. It appears that this course assignment impacted at least two students, since they took internships they applied for as part of this assignment. However, it is unknown if these students would have applied anyway.

Table 1: Students' Destinations

ID	Major	Internship or Full-Time	Company
1	Biology	Full-Time	*Still Looking*
2	CS	Internship (*)	HDR
3	CS	Internship	Microsoft
4	CS	Full-Time	Fisher Investments
5	CS	Internship	Cambia Health Solutions (+)
6	CS	Full-Time	Thermo Fischer Scientific (+)
7	Biology	Internship	Providence Cancer Research (+)
8	EE	Internship	EE consulting company
9	CS	Full-time	Government agency
10	CS	Internship (*)	Oregon Health Sciences Univ (+)
11	EE	Internship	Leidos
12	CS	Full-time	*Still Looking*

4.2 Q2: Résumé Content and Learning Outcomes

Content on students' final computational biology résumé was analyzed for themes related to the seven course learning outcomes:

1. LO1: Create and extend code to analyze and model biological data
2. LO2: Use databases to find biological data
3. LO3: Describe the functions of DNA, RNA, genes, transcription factors, and proteins
4. LO4: Analyze, trace, and execute computational biology algorithms through code and pseudo-code
5. LO5: Use software tools to analyze and interpret biological data
6. LO6: Synthesize, create, and communicate information related to biology and computation
7. LO7: Demonstrate planning, implementation, research, testing, analysis, teamwork skills, and communication skills

Table 2 shows the learning outcomes that appear in v3 of the computational biology résumés. Students 1, 6, and 12 did not submit résumé v3, so they are not included in Table 2. Students worked on semester-long, team projects with biology and biochemistry faculty members. As expected, all nine submissions included the computational biology course project on the final résumé. The development of new code or extending code (LO1) appeared on all final résumé submissions. Tracing and execution of algorithms (LO4) appeared on all submissions. Using software tools (LO5), such as BLAST, Jalview, and Cluster 3.0 appeared on seven of the nine submissions. While the project was done in teams, only six of the nine students explicitly mentioned teamwork or collaboration (part of LO7) with regard to the computational biology project, shown in the right-most column of the Table. Some part of LO7 was addressed in the project description on all nine submissions. Just one student explicitly mentioned communication skills (LO6) in the form of developing reports, demos, and presentations, which were required project deliverables. One student included using databases to find biological data (LO2). One student included biotechnology systems (LO3) on the résumé.

Table 2: Students' Final Résumés and Learning Outcomes

ID	LO1	LO2	LO3	LO4	LO5	LO6	LO7	Teamwork
2	x	x		x	x		x	x
3	x			x	x		x	x
4	x			x	x		x	
5	x			x	x		x	x
7	x			x	x		x	x
8	x			x	x		x	x
9	x			x	x		x	
10	x		x	x			x	
11	x			x		x	x	x

To demonstrate how the coding was completed, here are example résumé excerpts showcasing each learning outcome.

1. Developed a Python Program that takes a collection of FASTA files and automates BLAST searches on these files, as well as marking samples that contain certain organisms [student 9]
2. Made use of various genome databases to compare the yeast, fruit fly, and human genome [student 2]
3. Understand the uses of RNA-sequencing and DNA microarrays [student 10]
4. Proficient with String Algorithms, Sequence Evolution and Motif Finding [student 7]
5. Used Google's AlphaFold to predict structures of unknown proteins [student 4]
6. Communicated with research professor and group members to deliver project results, which were composed of Python functions, summaries, and other deliverables consistently [student 11]
7. Collaborated with interdisciplinary teams to extract insights from large-scale datasets. [student 3]

The instructor was curious about which tools appeared on the résumés, since the course primarily used python (biopython as a main library), some

R, and several specific computational biology tools. The tools that appeared on students' résumés included Python (9 of 9), BLAST (7 of 9), R (3 of 9), AlphaFold (2 of 9), Clustal Omega (2 of 9), Jalview (2 of 9), JavaTreeView (2 of 9), and Cluster 3.0 (1 of 9). All the software tools used in course activities appeared in at least one résumé. Otherwise, the frequency of appearance on résumés matched the frequency of tool use in the course.

4.3 Students' Reflections

Eight students earned extra credit (less than one percent of the overall grade) by completing an end-of-semester, online survey about all course deliverables. The survey study was approved by the Institution's Review Board.

Three survey questions asked about the résumé assignment. Students rated the statement: "The résumé and job application assignment *supported my learning* of computational biology course material." Of the eight students, two ranked "not descriptive", two ranked "minimally descriptive", two ranked "somewhat descriptive", one ranked "mostly descriptive", and one ranked "very descriptive". The instructor did not intend for this assignment to directly support student learning, but rather document what students experienced. It is not surprising that the rankings varied across all choices. Students ranked the other course activities (prelabs, labs, biotech briefing, project, exams, lectures) higher than the résumé assignment in terms of supporting learning.

Students rated the statement "The résumé and job application assignment *accurately demonstrated my understanding* of computational biology course material." Of the same eight students, one ranked "not descriptive", three ranked "minimally descriptive", one ranked "somewhat descriptive", two ranked "mostly descriptive", and one ranked "very descriptive". The assessment question had a slightly more positive overall ranking than supporting learning.

Students wrote free text to the following: "How did creating a computational biology version of your résumé impact your consideration of the computational biology field as a future endeavor? (no impact is a fine answer)". Of the eight students, five stated that it did not have much impact. Three said it had some impact. One said, "This task helped me to critically think about how the knowledge I have attained will benefit me beyond this course. It also helped me decide between data science and bioinformatics as career choices." [student 10] Another student stated, "It made some impact. A lot of the python focused parts and biological algorithms were interesting and were good to put on my resume." [student 11] Finally, a third student stated, "By creating a biology focused resume, it allowed me to see specifically where I applied those skills and made it easier for me to find internships that required these skills." [student 3] Even though the assignment appears to have minimal impact, overall, it assisted some students consider future pursuits.

5 Conclusions and Recommendations

The résumé assignment was designed to encourage students to reflect upon and document computational biology knowledge and skills during one semester. Four of seven learning outcomes were documented on the majority of final submissions. The job application deliverable required students to look for positions related to computational biology. By doing this assignment, students researched a variety of companies/organizations and reviewed the knowledge and skills that employers were seeking. Even though the assignment did not directly support learning new skills in computational biology, the assignment provided value for students in terms of career readiness. Creating a formal résumé was a new process for two students. Having this assignment opened conversations between the instructor and students about the variety of career pathways in computational biology. Students helped each other by sharing job advertisements with classmates and often asked the instructor to post advertisements to the course website.

This assignment can be used in any college course, especially in disciplines that have a direct tie to a profession, such as engineering, nursing, business, and education, where internships and co-op training are common co-curricular opportunities. The instructor did not provide specific guidelines regarding the length and appearance of the résumé. Some students made a one-page résumé, so they kept the information about the computational biology course very brief. Some students created longer résumés to include more detail about the course. Instructors who adopt this assignment may want to establish a résumé page limit that is consistent with their profession for entry-level jobs and internships. In addition, the instructor may want to refer students to the campus career center and provide links to online resources for how to prepare a résumé. For this study, the instructor had four résumé deliverables (original, v1, v2, and v3). Having just the original, v1, and v3 versions is probably sufficient. The instructor intentionally did not remind students of the learning outcomes when presenting the résumé assignment, so that students were not guided in terms of what to include. Instructors may want to have students annotate the mapping of resume content to learning outcomes, so they practice meta-cognition [8] by reflecting on how they achieved the goals of the course.

References

[1] ABET. "Criteria for Accrediting Computing Programs, 2023 – 2024". In: *ABET Accreditation* (2023). URL: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2023-2024/.

[2] Catherine G. P. Berdanier, Mary McCall, and Gracemarie Mike Fillenwarth. “Characterizing Disciplinarity and Conventions in Engineering Resume Profiles”. In: *IEEE Transactions on Professional Communication* 64.4 (2021), pp. 390–406. DOI: 10.1109/TPC.2021.3110397.

[3] Catherine G. P. Berdanier, Mary McCall, and Gracemarie Mike. “Résumés in the development of undergraduate engineering identity: A genre analysis with teaching implications”. In: *2016 IEEE International Professional Communication Conference (IPCC)*. 2016, pp. 1–9. DOI: 10.1109/IPCC.2016.7740488.

[4] The Joint Task Force on Computer Science Curricula. *Computer Science Curricula 2023*. 2023. URL: https://csed.acm.org/final-report/.

[5] Kerri A. Crowne et al. “A program for embedding career activities in multiple core business courses”. In: *International Journal of Management Education* 18.3 (2020). DOI: https://doi.org/10.1016/j.ijme.2020.100421.

[6] Murali Mani and Quamrul Mazumder. “Incorporating metacognition into learning”. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE ’13. Denver, Colorado, USA: Association for Computing Machinery, 2013, pp. 53–58. ISBN: 9781450318686. DOI: 10.1145/2445196.2445218. URL: https://doi.org/10.1145/2445196.2445218.

[7] Corbin Petersheim et al. “Comparing Student and Recruiter Evaluations of Computer Science Resumes”. In: *IEEE Transactions on Education* 66.2 (2023), pp. 130–138. DOI: 10.1109/TE.2022.3199685.

[8] MIT Teaching and Learning Lab. “Metacognition”. In: *Teaching Resources* (2024). URL: https://tll.mit.edu/teaching-resources/how-people-learn/metacognition/.

[9] Sreevani Thummaloor and Suresh Mutyala. “Improving resume writing skills of the final year undergraduates”. In: *AIP Conference Proceedings* 2794.1 (Oct. 2023), p. 020051. ISSN: 0094-243X. DOI: 10.1063/5.0165704. eprint: https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/5.0165704/18153302/020051_1_5.0165704.pdf. URL: https://doi.org/10.1063/5.0165704.

[10] Valerie K. Tucci, Abby R. O’Connor, and Lynn M. Bradley. “A Three-Year Chemistry Seminar Program Focusing on Career Development Skills”. In: *Journal of Chemical Education* 91.12 (2014), pp. 2071–2077. DOI: 10.1021/ed400667q. eprint: https://doi.org/10.1021/ed400667q. URL: https://doi.org/10.1021/ed400667q.

Undergraduate Perceptions on Attending Interdisciplinary Conferences[*]

Anna Ritz
Biology Department
Reed College
Portland, Oregon, 97202
aritz@reed.edu

Abstract

Attending computer science conferences can give students insight into the research process and how academic work is disseminated. This study examines undergraduate student perceptions about attending an interdisciplinary computational biology conference. The study was conducted over four academic years with a mix of participants who attended a conference as part of a course and participants who received an undergraduate travel award. Results from 70 students enrolled in nearly 30 different institutions indicate that attending conferences helped them learn about different careers, gave them a sense of what computational biology research entails, and provided insight into giving an effective oral presentation. We found that students who received a travel award felt more comfortable at the conferences than students who attended as part of a course. Based on these findings, we provide guidance about developing programs for undergraduate conference attendance.

1 Introduction

Computer science research is largely communicated through scientific meetings and conference proceedings. Thus, computer science conferences offer unique

[*]Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

opportunities for students to learn about the research process and the researchers themselves. Previous work has shown that undergraduates who have attended scientific conferences report increased confidence and an increased sense of belonging [15, 4, 5, 6], and guides on how to support undergraduates at conferences exist [2, 4, 8]. However, there have been few studies on how to best plan a conference experience for undergraduates.

As computer science matures as a field, interdisciplinary subfields have emerged. Computational biology has seen tremendous growth in the past few decades, and is a ripe area for undergraduates in biology and computer science to explore [12, 11, 13]. However, interdisciplinary research is rare at the undergraduate level [3], and undergraduate conference attendance is even rarer.

This paper examines undergraduate perceptions about attending an interdisciplinary computational biology conference. Many studies on student perceptions in conference attendance involve students who present research [5, 10], whereas our study includes students who attend conferences without presenting research. This work follows up on a pilot study about undergraduate conference attendance in 2016, which showed promise in integrating a conference experience in a course [7].

We surveyed two types of student experiences over four years: students who attended a conference as part of a course and students who attended a conference with a national travel award. We found that students who attended the conferences as part of a travel award had self-perceived better outcomes than students who attended the conferences as part of a course. In fact, whether a student was a travel awardee or a class participant was the *only* feature that produced statistically significantly different survey responses across demographic groups. This survey provides some guidance for developing programming around undergraduate conference attendance.

2 Conference Experiences and Survey Design

From 2018-2022, we offered two conference experiences to undergraduate students. In the **Class Model**, students enrolled in an upper-level computational biology class at Reed taught by the author and attended a conference as part of the course. Under this model, students attended an in-person conference or a virtual conference (all fully paid for) and completed scaffolded assignments designed to encourage them to network and engage in different aspects of the conference.

In the **Award Model**, the author organized a national undergraduate travel award in partnership with an annual computational biology conference. Undergraduates from any US-based institution were invited to apply and received conference registration and housing if they were selected. The applica-

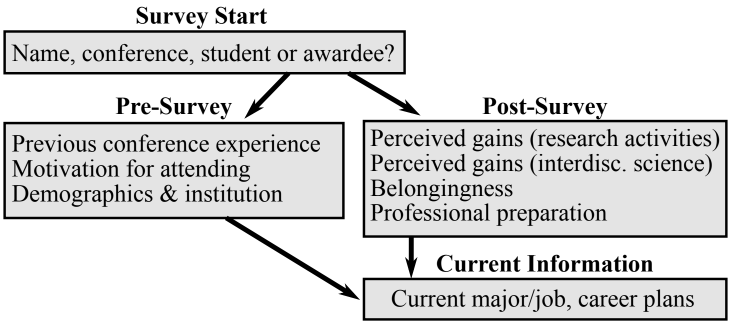

Survey Start
Name, conference, student or awardee?
Pre-Survey
Previous conference experience
Motivation for attending
Demographics & institution
Post-Survey
Perceived gains (research activities)
Perceived gains (interdisc. science)
Belongingness
Professional preparation
Current Information
Current major/job, career plans

Figure 1: Survey design. The data analyzed in this paper are for pre-post matched surveys.

tion was intentionally short, and we advertised the award to undergraduates at institutions within driving distance to the conference venue.

In both models, there was an emphasis on lowering the barrier for students to attend: any students at Reed with the prerequisites could register for the course and attend the conference, and the travel award prioritized students who hadn't previously attended a conference or had limited resources at their institution for computational biology. In both models, the author attended the conference, met with all the students, and held sessions for the students to get to know each other and learn about conference logistics [8].

2.1 Survey Design and Data Analysis

Students were prompted to complete a pre-survey before the conference, a post-survey immediately after the conference, and a post-survey annually thereafter. The full study design is available in the Supplementary Information S1 online.[1] for this paper, we focus on the matched pre-survey and post-survey that was completed within 6 months of the conference (Figure 1). The pre-survey asked respondents questions about their prior experience attending conferences, their reasons for attending, their demographics, and their institution and institution type (primarily undergraduate institution or research institution). The Likert-style post-survey questions were selected from Grinnell's Research on the Integrated Science Curriculum (RISC [9]) and CU Boulder's Undergraduate Research Student Self-Assessment (URSSA, [14]), with some additional questions specific to computational biology and interdisciplinary science. The full list of questions and aggregated responses are available as Supplementary Information S2 and S3 online.

[1] See the "Supplementary Information" section for the URL.

The post-survey contains 35 Likert-style questions. When comparing two groups of Likert-style responses, we use the Mann-Whitney U rank test which is used to analyze Likert data [1]. We used Python's `scipy` implementation of the test and used the Benjamini-Hochberg procedure for multiple hypothesis correction.

2.2 Predictions and Confounding Factors

We hypothesized that students in the Class Model would see larger self-perceived gains in research activities, interdisciplinary science, and belongingness due to the additional scaffolded assignments and the dedicated time for reflecting about the experience as a class. We also expected that student experiences would be different along demographic lines (e.g., gender, institution type, whether the student had previously attended a conference), but it was unclear which groups would see larger self-perceived gains.

There are several confounding factors that should be considered when interpreting the results of this study. First, there is a strong selection bias since the participants were not randomly chosen from a population of students. The computational biology course in the Class Model is an elective, so Reed students opt-in to take it. In the Award Model, students chose to apply by completing the travel award application. All students who participated in either model were invited to complete the survey. Second, the survey was administered between fall 2019 and spring 2023, which spanned the virtual/remote conferences during the COVID-19 pandemic. Three of the seven conferences in this study were virtual, which means that many of the answers to questions involving in-person events and networking may not be relevant. On the other hand, the low cost of virtual conferences provided opportunities for more students to attend these meetings. Finally, not all students who went to conferences responded to the survey. Students who had overall better conference experiences might be more willing to complete a follow-up survey years after the conference.

3 Survey Results and Discussion

One hundred and three students attended a conference between 2018-2022 (Figure 2). Seventy of these students (68%) took the survey at least once between fall 2019 and spring 2023. There were a total of 140 survey responses among these 70 individuals.[2] Fifty-two of the respondents (72%) completed the survey two or more times, usually in the immediate pre/post surveys (Supplementary Information S3 online).

[2]We record 71 respondents because one individual attended two different conferences.

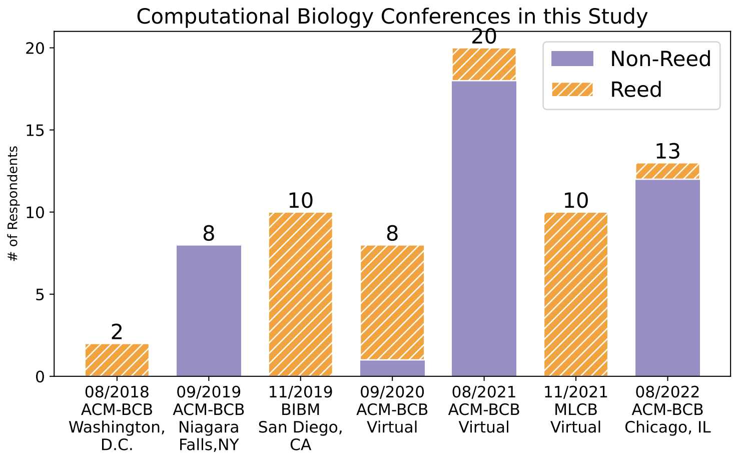


Figure 2: Computational biology conferences in this survey.

The respondents attended one of seven computational biology conferences (Figure 2). Most of the conferences were the ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM-BCB). One group of students attended the IEEE Conference on Bioinformatics and Biomedicine (BIBM) and another group of students attended the Machine Learning in Computational Biology (MLCB), which grew out of a NeurIPS workshop. The travel award was administered for ACM-BCB 2019, ACM-BCB 2021, and ACM-BCB 2022, whereas the Reed course took students to BIBM 2019, ACM-BCB 2020, and MLCB 2021. The ACM-BCB 2018 conference was a pilot study.

The 70 respondents came from 28 different institutions. Fifty-three (76%) of the respondents were from primarily undergraduate institutions (PUIs), with 31 (44%) from Reed. The remaining 17 were from research institutions. Twenty-six (37%) respondents participated for Reed course credit and 44 (63%) received a travel award (Figure 2). Fifty-two (74%) of the students had never attended a national conference, and thirty-eight (54%) had never attended a poster session or regional conference. Only nine of the respondents presented work at the conferences they attended. Of the individuals who chose to disclose their gender, 33 respondents were gender minorities (47%) and 25 respondents were male (36%). Of the individuals who chose to disclose their race/ethnicity, 32 respondents were White (46%), 19 respondents were Asian (27%), two respondents were Black or African American (3%), and four were multi-racial (6%). Additionally, three respondents were Hispanic/Latino (4%).

3.1 Post-survey trends confirm prior research

We examined the post-surveys for the 47 respondents who completed the survey within six months of the conference being held (Supplementary Information S4 online). Overall, students felt that they gained some experience in making and giving oral presentations (median Likert value 4), which is expected in a conference setting. Students strongly felt that the experience gave them a better sense of what computational biology research entails (median Likert value 5 and no score below 3) and they learned about different scientific careers (median Likert value 4). Regarding questions surrounding conference dynamics and belongingness, the respondents overall felt that the conferences were welcoming, the talks were relevant to their interests, the experience helped them clarify career path, and that it made them more likely to try research (median Likert value 4). Many of these perceived benefits have been noted elsewhere [15, 4, 5, 6]. All 70 respondents indicated they would want to attend another conference in the future. However, some responses were not as positive: students felt that the conference assumed too much prerequisite knowledge (median Likert 4), and there was high variance in whether the respondents felt out of place at the conference.

3.2 Differences between the Class and Award Model participants

We then measured the differences in responses between the 26 students who attended a conference under the Class Model and the 44 students who attend the conference under the Award Model (Supplementary Information S5 online). There were two statistically significant differences between the two groups (Figure 3). First, students in the Class Model felt that the conferences were *less* welcoming than travel awardees (Mann-Whitney U corrected p-value 0.007). Second, students in the Class Model felt *less* comfortable talking with other attendees in a professional networking session (Mann-Whitney U corrected p-value 0.028). Responses to other questions showed similar trends: students in the Class Model felt slightly less confident in their ability to contribute to science, and the conference made them slightly less excited about research and less likely to try research (Figure 3).

These results suggest that the students who attended under the Award Model felt more comfortable at the conferences than students who attended under the Class Model. There are many differences between these two groups, but four potentially contributing factors stick out. First is the motivation for attending the conference in the first place. The students who received a travel award had the interest to apply, whereas the Class Model group opted in by simply enrolling in the class. Second is the sense of community: students who received a travel award met at the conference in an initial awardee dinner,

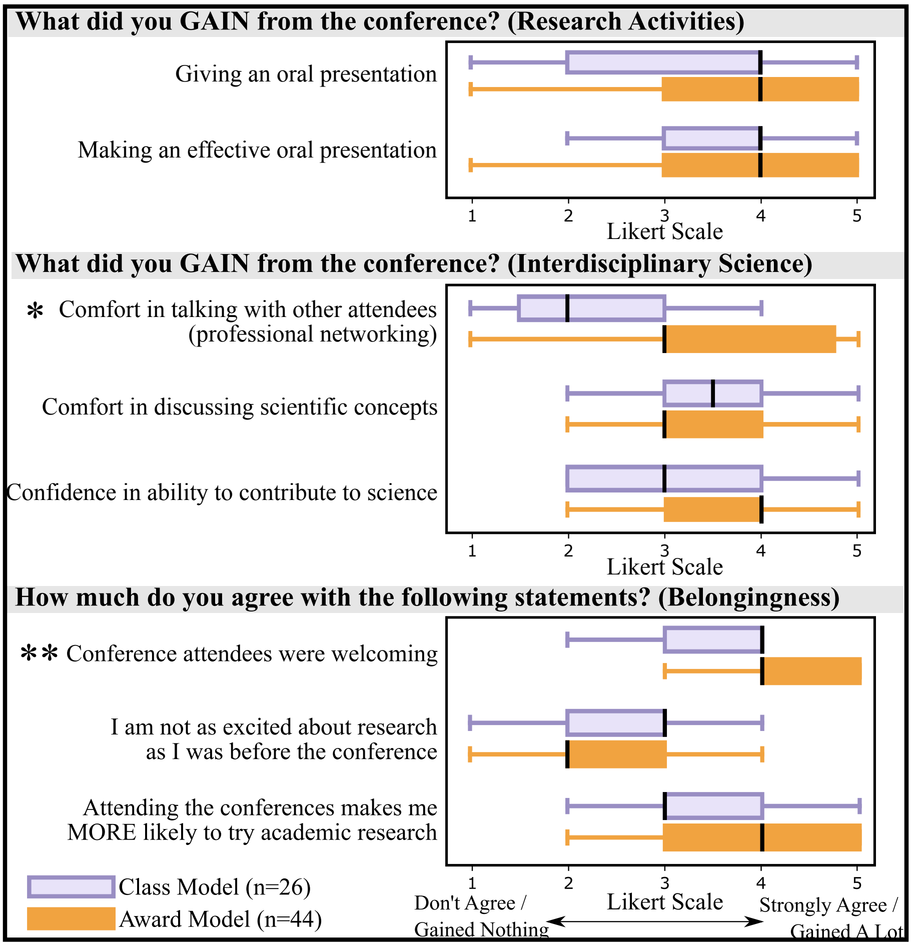


Figure 3: Select responses for students with course credit vs. travel award. Boxes indicate median and quartile ranges and the whiskers indicate 1.5 times the interquartile range. ∗: Mann-Whitney U corrected p-value$<$ 0.05; ∗∗: corrected p-value$<$ 0.01.

which offered built-in networking with a new group of people before the conference even began. Third, the two groups prepared very differently: travel awardees received information specific to the conference and attended new attendee sessions. Students in the course, on the other hand, had scaffolded assignments to prepare them for the meeting (and ultimately earned a grade in the course). Finally, the response rate for the two groups is also dramatically different. Students in the Class Model had a 100% response rate, since they saw the instructor after the conference and were given many in person reminders. Travel awardees did not have a similar pressure to complete the survey, so those who would have reported negative experiences may have chosen to not participate in the survey.

3.3 Demographic groups did not differ in responses

While our study focused on evaluating the differences between the Class Model and the Award Model, it is possible that a response may be due to a student's experience and background (such as prior interests, demographics, and motivation for attending). We found no significant differences in responses for gender minorities vs. men, White students vs. non-White students, students from PUIs vs. students from R1 institutions, and students who had previously attended a conference vs. students who had never attended a conference before (Supplementary Information S5 online). Some of the groups have small sample sizes; however, these results are reassuring in that students with different preparation, affiliations, and interests generally had similar conference experiences, according to the responses.

4 Conclusions and Recommendations

This paper presents a four-year survey about undergraduate perceptions of conference attendance, with respondents representing a broad range of undergraduates by institution type, gender, race, and motivation for attending a conference. Based on the survey responses, students who received a travel award felt more comfortable at the conferences than students who attended as part of a course. While it is challenging to draw additional conclusions from data, attending the conferences clearly helped some respondents clarify their career paths and gain a better understanding of computational biology as an interdisciplinary field.

The Award Model may foster belongingness in a way that is absent from the Class Model. Students who apply for and receive travel funds might already feel more included in the community, even if the travel award process is quite short. The Award Model also sets different expectations than the Class Model: travel awardees met a dozen other students from different schools in a professional setting, compared to traveling to a meeting with classmates. Tying the conference experience to a course grade in the Class Model may have decreased the motivation of students to explore options on their own; more work could be done to improve the scaffolded conference experience in the classroom to make students feel better prepared for the conference. This study suggests that developing a program that incentivizes students to opt-in to conference attendance will be a good first step towards community building through conference experiences. For example, providing department funds for students to attend conferences might be a better approach than offering it as a class assignment.

Some limitations of the study makes it hard to understand the impact of conference attendance. The survey included optional open-ended responses, but did not provide enough information to compare the two models properly (Supplementary Information S6 online). There was a longitudinal component to the survey to determine how conference attendance might impact career choice, but again there was limited data to draw larger inferences. Disaggregating the survey results by virtual vs. in-person conferences may help untangle the benefits of those experiences, specifically the balance between professional development opportunities for in-person conferences and the lower cost more accessible environment for virtual conferences.

Regardless of the model, preparing students through pre-conference sessions and helping them navigate travel or professional networking is critical to help them feel like they belong at the conference. Connecting undergraduates with each other early on in the conference helps them navigate the experience with peers. Finally, it is important to encourage students early in their time in college to take advantage of these opportunities, even before they might have had formal research experience.

Supplementary Information: Information about the full study design, survey questions, aggregated responses, and a full set of all results can be found as Supplementary Information S1-S6 online at `https://www.reed.edu/biology/ritz/ccsc-supplementary-info.html`.

Acknowledgments: This work was supported by NSF-DBI-1750981. The project was deemed exempt by Reed's IRB in April 2018 and remains active as of July 2024. We thank the students who participated in the conferences and in the follow up surveys.

References

[1] Dane Bertram. "Likert scales". In: *Retrieved November* 2.10 (2007), pp. 1–10.

[2] Janet Davis and Christine Alvarado. "Supporting undergraduates to make the most of conferences". In: *ACM Inroads* 8.3 (2017), pp. 32–35.

[3] S. N. Davis et al. "Mentoring Undergraduate Scholars: A Pathway to Interdisciplinary Research?" In: *Mentoring & Tutoring: Partnership in Learning* (2015), pp. 1–14.

[4] Elizabeth A Flaherty, Rachael E Urbanek, Darren M Wood, et al. “A Framework for Mentoring Students Attending Their First Professional Conference”. In: *Natural Sciences Education* 47.1 (2018).

[5] Herbert W Helm and Karl GD Bailey. “Perceived benefits of presenting undergraduate research at a professional conference.” In: *North American Journal of Psychology* 15.3 (2013).

[6] Anne-Barrie Hunter, Sandra L Laursen, and Elaine Seymour. “Becoming a scientist: The role of undergraduate research in students’ cognitive, personal, and professional development”. In: *Science education* 91.1 (2007), pp. 36–74.

[7] Amy R Lazarte and Anna Ritz. “Lowering the Barrier for Undergraduates to Learn about Computational Research through a Course-Based Conference Experience”. In: *2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT).* Vol. 1. IEEE. 2020, pp. 1–4.

[8] Elizabeth Leininger et al. “Ten simple rules for attending your first conference”. In: *PLOS Computational Biology* 17.7 (2021), e1009133.

[9] David Lopatto. *RISC Survey.* `https://sure.sites.grinnell.edu/risc-survey/`. Accessed: 2023-08-01.

[10] Patricia Ann Mabrouk. “Survey study investigating the significance of conference participation to undergraduate research students”. In: *Journal of Chemical Education* 86.11 (2009), p. 1335.

[11] Nicola Mulder et al. “The development and application of bioinformatics core competencies to improve bioinformatics training and education”. In: *PLoS computational biology* 14.2 (2018), e1005772.

[12] Layla Oesper and Anya Vostinar. “Expanding undergraduate exposure to computer science subfields: Resources and lessons from a hands-on computational biology workshop”. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education.* 2020, pp. 1214–1219.

[13] Lonnie Welch et al. “Bioinformatics curriculum guidelines: toward a definition of core competencies”. In: *PLOS computational biology* 10.3 (2014), e1003496.

[14] Timothy J Weston and Sandra L Laursen. “The undergraduate research student self-assessment (URSSA): Validation for use in program evaluation”. In: *CBE?Life Sciences Education* 14.3 (2015), ar33.

[15] Heather M Wright and N Burçin Tamer. “Can Sending First and Second Year Computing Students to Technical Conferences Help Retention?” In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* ACM. 2019, pp. 56–62.

Student-perspective Observations from the Comparison of Rust and C++ Languages*

Josiah Scott, Fei Zuo, Junghwan Rhee
Department of Computer Science
University of Central Oklahoma, Edmond, OK 73034
{jscott70,fzuo,jrhee2}@uco.edu

Abstract

Rust language has been widely adopted in industry, government agencies, and education sectors because of its secure features that reduce memory errors while maintaining fast performance. However, many first-time learners including undergraduate students experience a high learning curve in this language compared to other programming languages such as C++, Java, and Python. In this paper, we present a student-driven study to compare the code of the implementation of a webserver written in two different languages, C++ and Rust. Specifically, we present our observations with the pairs of code snippets written in C++ and Rust regarding what are the major differences highlighting the main factors causing difficulty for new learners.

1 Introduction

Rust programming language has gained significant traction in recent years, across various sectors because of its focus on memory safety and performance [4]. Despite the advantages the language offers, it is often regarded as having a steeper learning curve than more established languages like C++, Java, or Python. Learning the Rust language is beneficial to students, but we would

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

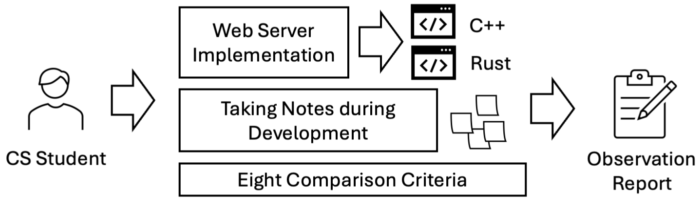

Web Server Implementation
C++
Rust
CS Student
Taking Notes during Development
Eight Comparison Criteria
Observation Report

Figure 1: Process of a Comparison Study.

like to find what are the major differences and roadblocks perceived by undergraduate students so that we can design an effective course in the future and such studies need further work.

This paper aims to compare Rust with another widely used programming language to identify and analyze language design features and pieces of the development experience that provide insight into the learning challenges and benefits of Rust. To do this, we have a study where an undergraduate student writes pairs of code snippets in Rust and C++ that offer the same functionality and evaluate the differences between the two languages.

The contribution of this paper includes the following:

- We conduct a study from an undergraduate student's point of view regarding the experience of writing code to implement the same functionality in two different languages.
- From this study we extracted a set of observations by the student regarding the different characteristics of two languages that give us insight on how we need to design our future Rust course.

Section 2 presents the design of our comparison study. Section 3 presents the comparison analysis of C++ and Rust code. Our discussion is presented in Section 4. Related work is discussed in Section 5. Finally, this paper is concluded in Section 6.

2 Design of a Comparison Study

Figure 1 presents the process of our comparison study. A computer science undergraduate student implemented a program in two different languages, Rust and another one for comparison, performing the same functionality. For this comparison, C++ was chosen as the contrasting language due to its widespread use in education and industry, as well as its non-functional similarities to Rust, such as common use cases and performance. To systematically compare Rust and C++ from a learner's perspective, we selected key areas of difference where new developers to each language may encounter difficulties. We selected the development of a simple web server as the comparison medium. A web server was chosen because it encompasses a wide range of programming concepts, and is a reasonable use case for both languages. The web servers have the same basic structure and features, with request parsing, response generation, file serving, and routing.

During and after development, notes for complications and language differences were taken and used to compile a list of areas for comparison. These areas are (1) the compilation and run process, (2) mutability, (3) control flow, (4) error handling, (5) input and output, (6) memory references and borrowing, (7) concurrency with threads, and (8) compiler messages. From this list, we compiled code snippets that illustrate the differences between the languages in these areas. These code snippets are compared and evaluated based on ease of use, learning curve, readability, and safety.

3 Comparison Analysis of C++ and Rust Code

In this section, we present our comparison of web server code written in C++ and Rust languages regarding eight different areas.

3.1 Compilation and Execution Process

Rust provides a much simpler compilation and run step. Thus learning Rust does not need to know anything about linkers, object files, or manual installation of custom libraries. Similar to Python's `pip`, Rust comes with a package manager, `cargo`, installed out of the box. `Cargo` allows for easy installation of dependencies and a simple `run` command. This is not unique to Rust, but is certainly an advantage in terms of ease of use when compared to C++.

```
g++ main.cpp  -I"C:\Program Files\Boost\
boost_1_85_0" -L"C:\Program Files\Boost\
boost_1_85_0\stage\libs" -lws2_32

./a.exe
```

Figure 2: C++ build and run process.

```
cargo run
```

Figure 3: Rust build and run process.

Figure 2 shows two steps to build C++ code and execute it. In comparison, Rust provides a supporting program to simplify the build process and execution called "`cargo`" shown in Figure 3.

Cargo is especially useful for integrating third-party modules/libraries that further have their own dependencies.

***Observation** #1: Rust integrates software component management and runtime tools together with the compiler.*

3.2 Mutability

Most programming languages have support for immutable variables, like with the `const` keyword in C++. Variables in Rust however, are immutable by default. This encourages a style of programming that reduces unintended modifications and bugs. Because immutability by default is not the norm in most languages including C++, this is a feature that may cause confusion among learners for a short time until they get familiar with it.

```
std::string response = "Hello, World!";
response = "New response";
```

Figure 4: C++ code.

```
let response = "Hello, World!";
//response = "New response";

let mut response = "Hello, World!";
response = "New response"; //This is allowed
```

Figure 5: Rust code.

Figure 4 shows no restriction of an assignment of a new string in C++. Figure 5 shows that an auto variable will not allow a new assignment while an explicit mutable ("`mut`") string can be overwritten.

***Observation** #2: Rust prevents unintended modifications by explicitly expressing mutability, which helps developers think about the mutability of variables.*

3.3 Control Flow and Pattern Matching

C++ has traditional conditionals for control flow, while Rust has these same conditionals as well as pattern matching. Pattern matching using the `"match"` keyword allows you to compare a value against a series of patterns and execute code based on which pattern matches. This feature is powerful but the syntax and the idea of matching patterns instead of evaluating conditions is harder to grasp.

C++ will need to utilize nested if and else statements for multiple patterns (Figure 6). Rust offers a clean choice for multiple patterns as shown in Figure 7. As we have a higher number of matching patterns, Rust's syntax will have a higher advantage showing lower complexity with concise code.

```cpp
if (routes.find(url) != routes.end()){
    // Handle route...
}
else {
    // Handle 404...
}
```

Figure 6: C++ code.

```rust
match routes.get(url) {
    Some(route) => {
        // Handle route...
    }
    None => {
        // Handle 404...
    }
}
```

Figure 7: Rust code.

***Observation** #3: Rust's pattern−matching structure and syntax provide convenience to avoid cumbersome nested if−else statements.*

3.4 Error Handling

C++ uses exception-based error handling, which can be caught using `try` and `catch` blocks. Rust uses the "`Result`" type which enforces error handling at compile time. This makes for more robust code but adds to the learning curve of Rust.

```cpp
try{
    std::string content =
        read_html_file("path/to/file");
} catch (const std::exception&) {
    std::cerr << "Error: " <<
        e.what() << std::endl;
}
```

Figure 8: C++ code.

```rust
match read_html_file("path/to/file"){
    Ok(content) =>
        println!("File content: {}", content),
    Err(e) => println!("Error: {}", e),


}
```

Figure 9: Rust code.

Figure 8 shows a basic example of error handling in C++ using `try` and `catch` keywords. Figure 9 illustrates error handling in Rust using the "`Result`" type. The "`read_html_file()`" function returns a Result which can either be "`Ok`" if successful, or "`Err`" if the function has any errors. Pattern matching is then used to handle both cases.

***Observation** #4: Rust provides clear pattern−matching syntax and Ok/Err syntax to support error handling concisely in the language level.*

3.5 Input and Output, File Handling

C++ reads files using `std::ifstream` and `std::stringstream`, which are straightforward for C++ developers. The concepts of streams and buffers are being used less and less in modern programming languages, so this approach may be unfamiliar to many developers. Rust uses more straightforward "`open`" and "`read_to_string`" functions. Rust's approach returns a `Result` type, incorporating error handling into the function signature.

```cpp
std::string read_html_file(
        const std::string &path){
    std::ifstream file(path);
    std::stringstream buffer;
    buffer << file.rdbuf();
    return buffer.str();
}
```

Figure 10: C++ code.

```rust
fn read_html_file(path: &str) ->
        Result<String, std::io::Error> {
    let mut file = File::open(path)?;
    let mut contents: String = String::new();
    file.read_to_string(&mut contents)?;
    Ok(contents)
}
```

Figure 11: Rust code.

Figure 10 shows C++ code for a file input using the stream insertion operator and a buffer. Figure 11 shows Rust instead using a `File` type and simple functions on that type.

***Observation** #5: Rust provides an explicit way to express "read a file to a string" using its standard library API.*

3.6 Memory References and Borrowing

Rust and C++ have similar syntax for taking memory allocated in scope and utilizing it in another scope (the `&` operator in function headers), but the rules surrounding this concept is very different across the languages. C++ has references (and pointers) that allow programs to reference the same memory across scopes without copying it. Rust instead has borrowing, which has strict rules and compile time checks to make sure a reference is always valid. This prevents issues like dangling references, but the strict rules can be confusing. In Rust each value has a single owner at a time. Ownership can be transferred or temporarily borrowed.

```cpp
void accept_connections(
    io_context &io_context,
    tcp::acceptor &acceptor){
    // Function body
}
```

Figure 12: C++ code.

```rust
fn handle_request(filepath: &str,
    mut stream: TcpStream) {
    // Function body

}
```

Figure 13: Rust code.

Figure 12 demonstrates passing references in C++. The "`io_context`" and "`acceptor`" are passed by reference, allowing the body of the function to modify the original objects. Figure 13 shows borrowing in Rust. The "`filepath`" is borrowed immutably and the "`stream`" is borrowed mutably.

***Observation** #6: Rust's ownership and borrowing are one of unfamiliar and cumbersome concepts, however they are helpful to defeat memory errors.*

3.7 Concurrency with Threads

Both Rust and C++ provide support for concurrency with threads in their standard library. C++ uses a relatively simple function to spawn and move an object to the new thread. Rust on the other hand uses a spawn function that accepts a closure, and requires a special "`move`" keyword to transfer ownership of the captured variables into the new thread. This requires a solid grasp of closures, as well as Rust's ownership and borrowing principles.

```
void accept_connections(
    io_context &io_context,
    tcp::acceptor &acceptor){
  while (true){
    tcp::socket socket(io_context);
    acceptor.accept(socket);
    std::thread(handle_request,
        std::move(socket)).detach();
  }
}
```

Figure 14: C++ code.

```
fn main() {
  let routes = create_routes();
  let listener =
    TcpListener::bin("127.0.0.1:3000").unwrap();
  for stream: Result<{unknown}, {unknown}>
      in listener.incoming() {
    match stream{
      Ok(stream) => {
        let routes = routes.clone();
        thread::spawn(move || {
          handle_client(stream, &routes);});
      }
      // Handle Error
    }
  }
}
```

Figure 15: Rust code.

Figure 14 demonstrates thread creation in C++ with the "`move`" function moving the TCP socket to the new thread, and the "`detach`" function which allows the thread to execute independently. Figure 15 shows thread creation in Rust, in which a variable must be cloned, captured by a closure, and have its ownership transferred into the closure.

***Observation** #7: Concurrency with threading in Rust requires knowledge of more advanced programming concepts.*

3.8 Compiler Messages

Compiler error messages exhibit a disparity in clarity, relevance, and helpfulness between Rust and C++. The Rust compiler is known for having clear and helpful compiler error messages. The Rust compiler often gives specific line numbers, detailed explanations of the mistake, and suggests solutions. C++ compiler messages are often less clear and require more specialized knowledge to decipher.

```
std::ifstream file(path);

incomplete type "std::ifstream" is not allowed C/C++(70)
```

Figure 16: C++ code and an error message.

Figure 16 shows the code and the resulting error from attempting to create a variable of type "`ifstream`" without the necessary library import. The compiler error message simply says that "`ifstream`" is not allowed.

```
let mut file = File::open(path)?;

error[E0433]: failed to resolve: use of undeclared type `File`
  --> src\main.rs:53:20
   |
53 |     let mut file = File::open(path)?;
   |                    ^^^^ use of undeclared type `File`
   |
help: consider importing this struct
   |
1  + use std::fs::File;
   |
```

Figure 17: Rust code and an error message.

In contrast for the same mistake, Figure 17 shows Rust giving a much more detailed and helpful compiler message, even suggesting that the correct import statement is necessary.

Observation #8: *Rust offers more detailed and context−rich error messages that are helpful for debugging.*

4 Discussion

In our analysis, we identified several key differences between Rust and C++ that impact the learning experience. Rust's default immutability, and enforced error handling contribute to safer and more robust code but require learning more concepts up front to use the language. The same is true for Rust's ownership and borrowing based memory management. Rust's helpful compiler aids in reducing confusion in debugging situations, but is not perfect. The `Cargo` tool for Rust also makes adding packages or libraries and running programs much easier than C++'s more manual approach.

5 Related Work

Coblenz et al. [2] analyzed Stack Overflow posts and found that users struggle with ownership, borrowing, standard library and the "try operator". They proposed some solutions for these struggles in the form of potential tools that could be created. Likewise, Zhu et al. [7] manually analyzed 100 Stack Overflow posts to find what safety features were difficult to understand and apply, and whether or not the compiler was helpful enough in debugging. They also did a survey with Rust developers to validate their findings. They concluded that ownership and lifetimes were the most difficult to understand and apply.

Zeng et al. [6] used Reddit and Hacker News posts to investigate why Rust is difficult for many to adopt. The study found that main factors are that Rust has good tooling that is not promoted enough, and the cost of migrating existing codebases is too high. The survey [5] performed in 2023 shows that the learning curve of Rust was still a common reason why developers have not started to learn the language. The top three areas of Rust that developers seem to struggle with are asynchronous programming, traits and generics, and the borrow checker. Abtahi et al. [1] observed how experienced developers learn Rust. They also proposed some ideas to make the learning process easier for package and language designers. These included providing ample example code, create inline compiler messages for packages, and linking error messages to explanations of the relevant concepts.

In the computing education context, Dixon [3] discussed the potential of Rust and believed that integrating Rust programming into operating system courses would benefit students.

6 Conclusion

In this paper, we conduct a comparative study by an undergraduate student to write a web server program in Rust and C++. We find Rust, as a new language, has multiple advanced and convenient functions, code structures, and syntax integrated into its design. While learning new functions, especially memory-related features takes a high learning curve, we found that it will be beneficial to implement a program with fewer bugs and less amount of code.

References

[1] Parastoo Abtahi and Griffin Dietz. “Learning Rust: How Experienced Programmers Leverage Resources to Learn a New Programming Language”. In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–8.

[2] Michael Coblenz et al. “A Multimodal Study of Challenges Using Rust”. In: *the 13th Annual Workshop at the Intersection of PL and HCI*. 2023.

[3] Bryan Dixon. “Position Paper on Teaching Operating Systems Using the Rust Programming Language”. In: *Journal of Computing Sciences in Colleges* 39.1 (2023), pp. 11–17.

[4] Ralf Jung et al. “Safe systems programming in Rust”. In: *Communications of the ACM* 64.4 (2021), pp. 144–152.

[5] The Rust Survey Team. *2023 Annual Rust Survey Results*. `https://blog.rust-lang.org/2024/02/19/2023-Rust-Annual-Survey-2023-results.html`. Accessed: 2024-06-25. 2024.

[6] Anna Zeng and Will Crichton. “Identifying Barriers to Adoption for Rust through Online Discourse”. In: *the 9th Workshop on Evaluation and Usability of Programming Languages and Tools*. 2018.

[7] Shuofei Zhu et al. “Learning and programming challenges of Rust: a mixed-methods study”. In: *the 44th International Conference on Software Engineering*. 2022, pp. 1269–1281.

The Effectiveness of Coding LLMs and the Challenges in Teaching CS1/2: A Case Study*

Alexander Hong[1] and Gongbing Hong[2]
[1]Computer Science
Duke University
Durham, NC 27708
alexander.hong@duke.edu
[2]Information Systems and Computer Science
Georgia College and State University
Milledgeville, GA 31061
gongbing.hong@gcsu.edu

Abstract

This paper presents a case study that evaluates the effectiveness of coding Large Language Models (LLMs) in introductory computer science courses at the university level. The study assesses six different AI-powered code generators. The evaluation focuses on the accuracy of these AI code generators in solving ten programming problems from a set of problems that instructors at Duke University can assign to students for weekly completion. The results demonstrate the effectiveness of coding LLMs in solving these problems.

Based on the findings, the paper discusses the challenges faced by the computer science education community and potential strategies to address them. The advent of coding LLMs poses significant challenges to traditional teaching and learning methods in computer science. These challenges include the need for strategies to mitigate any negative impact of LLMs on the learning process. At the same time, these code LLMs also offer tremendous opportunities for enhancing teaching and learning.

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Computer science education plays a fundamental role in shaping the programming skills and problem-solving abilities of future software developers. Traditionally, students have learned programming through a process of writing code, testing, and debugging their solutions to programming problems. This meticulous process helps students develop critical thinking skills, learn how to develop solutions, and enhance their coding abilities.

However, the advent of Artificial Intelligence (AI) and Machine Learning technologies has introduced new tools, such as Large Language Models (LLMs), that have gained popularity in the coding community for their potential to assist coders with repetitive tasks. These AI-powered coding assistants can generate code snippets, provide suggestions, and improve code quality. They can even generate complete solutions to many simple programming problems, including those commonly used as assignments in classrooms. The wide availability of these AI tools raises concerns about their impact on computer science education, the challenges they pose to educators, and the integrity of the educational process.

On the other hand, LLMs also present opportunities for educators to enhance the learning experience for students. By integrating LLMs into their curriculum, educators can provide additional support and resources to help students learn programming concepts more effectively.

In this paper, we present a case study evaluating the effectiveness of coding LLMs in university-level introductory computer science courses. We assess the accuracy of six AI-powered code generators in solving programming problems commonly used in introductory computer science courses. The results of the study provide insights into the performance of these LLMs in solving programming problems and their potential impact on computer science education. Additionally, we discuss the challenges faced by educators and students in the context of AI-powered coding assistants and potential strategies to address these challenges.

2 Background and Related Work

Recent advances in AI have led to the development of large language models (LLMs) that have demonstrated impressive performance in various domains, including code generation. This has raised concerns about the potential impact of LLMs on education, particularly in introductory programming courses.

Several studies have evaluated the ability of LLMs to solve programming problems. Wang et al. [14] conducted a study to assess the performance of ChatGPT on 187 problems from six undergraduate computer science courses.

They found that ChatGPT was able to solve around 61% of the problems, including multiple-choice, short-answer, and programming questions.

In another study, Denny et al. [3] examined the effectiveness of GitHub Copilot, an AI-powered code generator integrated into the Visual Studio Code (VSCode) editor, in solving 166 publicly available programming problems for CS1. They found that Copilot was able to solve half of the problems on its first try and 60% of the remaining problems with some additional prompting.

Denny et al. [4] also discussed the challenges and opportunities that computing educators and students face with the rise of coding LLMs. They argue that it is important to incorporate these tools into the curriculum from the beginning to help students learn how to use them effectively and responsibly.

Lau and Guo [7] conducted a study to investigate the perspectives of introductory programming instructors on the use of AI-powered code generators. They found that instructors were initially hesitant about the use of these tools but later shifted their focus to how to integrate them into their teaching practices.

Codio [2] conducted a survey of 371 students to understand their perspectives on the use of generative AI in education, including computing education. The survey found that the majority of students (76%) were aware of generative AI, and many (47%) had used it for school-related work. The survey also found that students generally care about academic integrity when using generative AI tools.

3 Methodology

3.1 Selection of the Programming Problems

In introductory computer science courses at Duke University, students complete weekly Algorithmic Problem-solving Testing (APT) problems assigned by their instructors. These problems assess students' understanding of programming concepts and problem-solving skills. The problems encompass a broad range of topics, including programming techniques, data structures, and algorithms. Examples of APT problems can be found in [12, 13]. Students are required to solve these problems using the programming language specified by the instructors, which is Python for CS1 and Java for CS2. Students submit their solutions to an internal online judging platform, where the uploaded code is automatically executed against a set of test cases. These test cases have been validated through years of use.

For this study, we selected ten programming problems from the APT set to evaluate the performance of coding LLMs in solving these problems. The problems were chosen to represent varying levels of difficulty and a variety of topics covered in introductory computer science courses. The problems are divided into two subsets (Table 1): five problems from CS101 (Introduction to Computer Science) and five problems from CS201 (Data Structures and Algorithms).

Table 1: Selected APT Problems

Problem	**Description**
	CS 101
ChangeMoney	Given exchange rates, convert one currency amount into another.
LengthMost	Given a phrase, calculate which word length occurs the most number of times.
PopularCategory	Calculate the most popular category of a group of words, returning a string of sorted and alphabetized words in that category.
SwapParts	Given a phrase, return a copy of the phrase with individual words in the same order but changed according to a set of predefined rules.
WordChoices	Given four phrases of words, determine which words fit with a given algorithm.
	CS 201
Aaagmnrs	Given a list of phrases, remove each phrase that is an anagram of an earlier phrase.
MergeLists	Takes two lists with the same number of nodes and weaves them together alternatingly.
SortedFreqs	Given a list of strings, calculate how frequently each unique string occurs.
Starter	Determine how many unique words in an array of words start with a specified letter.
StringCuts	Filter strings out of a list that satisfy a minimum acceptable length.

3.2 Code Generators

This study evaluates the performance of six AI-powered code generators in solving the selected programming problems. The six code generators are GitHub Copilot [5], OpenAI ChatGPT-3.5 [9], StarCoder [15], Meta Llama [8], Amazon CodeWhisperer [11], and Replit AI [10]. The versions used in the study and their training are listed in Table 2. These code generators were chosen based on their availability, popularity, and potential to assist students in completing programming assignments. The coding tools were tested in their default configurations without additional training or customization.

Table 2: Code Generator Details

Code Generator / Version	**Training**
GitHub Copilot / v1.140.0	Trained on a selection of the English language, public GitHub repositories, and other publicly available source code. [5]
ChatGPT-3.5 / v3.5	Trained using a large amount of text data from web pages, books, and articles. [6]
StarCoder / 15.5B	Trained on permissively licensed data from GitHub. [15]
Llama / 13b	Trained on web pages, open source repositories of source code, public domain books, scientific papers, Q&As from Stack Exchange. [8]
CodeWhisperer / 1.9	Trained on billions of lines of Amazon and publicly available code. [11]
Replit AI / 2023.10 (Initial)	Trained on publicly available code and tuned by Replit. [10]

Standard prompts were generated and executed *once* for all code generators to ensure that they did *not* learn or adapt to any specific prompts through reinforcement learning. All tests were conducted between September and November 2023.

3.3 Evaluation Metrics

The study assesses the performance of the code generators based on how accurately they can solve the selected programming problems. The evaluation is conducted by uploading the generated code to the same online judging platform used by Duke students. The accuracy of the code generators is measured as the percentage of test cases for which the generated code produces the correct output.

4 Results

4.1 Accuracy

Table 3 presents the accuracy of the code generators in solving the selected programming problems. The results indicate that ChatGPT-3.5 and Replit AI exhibited superior performance, both achieving an average accuracy rate of approximately 89%. Notably, both generators performed better when solving Python problems than Java problems. This disparity can be attributed to the underlying LLMs being trained on a larger volume of Python data compared to Java data. Copilot ranked as the third-best performer, attaining an average accuracy of 76.8%. For each programming problem included in the study, *at least* one code generator demonstrated the ability to solve the problem with 100% accuracy. This finding suggests that AI-powered code generators can effectively solve programming problems commonly encountered in introductory computer science courses.

Table 3: Accuracy (%) of Code Generators

Problem (Test Cases)	Copilot	ChatGPT	StarCoder	Llama	CodeWhisperer	Replit AI
CS101 Problem Set (Python)						
ChangeMoney (24)	100.0	100.0	100.0	100.0	100.0	100.0
LengthMost (20)	75.0	100.0	0.0	0.0	50.0	100.0
PopularCategory (23)	56.5	100.0	0.0	0.0	4.3	100.0
SwapParts (27)	100.0	100.0	0.0	3.7	11.1	100.0
WordChoices (27)	96.3	100.0	0.0	14.8	3.7	100.0
CS201 Problem Set (Java)						
Aaagmnrs (50)	0.0	100.0	14.0	8.0	4.0	14.0
MergeLists (8)	100.0	25.0	0.0	0.0	0.0	100.0
SortedFreqs (13)	69.2	100.0	100.0	30.8	100.0	100.0
Starter (14)	71.4	71.4	100.0	71.4	71.4	71.4
StringCuts (10)	100.0	100.0	70.0	30.0	70.0	100.0
Average Accuracy % (Stddev %)						
CS101 (Python)	85.6 / 19	100.0 / 0	20.0 / 45	23.7 / 43	33.8 / 42	100.0 / 0
CS201 (Java)	68.1 / 41	79.3 / 33	56.8 / 47	28.0 / 28	49.1 / 45	77.1 / 37
Overall	76.8 / 32	89.6 / 24	38.4 / 48	25.9 / 34	41.5 / 41	88.5 / 28

4.2 Experience on Usability and Speed

All code generators under study, except ChatGPT, are accessible as plug-ins within code editors such as VSCode. Users can effortlessly enable and disable the code generators within the editor, except for StarCoder and Llama. Once the plug-in is enabled, users can prompt the code generators by simply writing blocks of comments in the editor. As users continue editing their code, the code generators, based on the current context, often provide suggestions in the form of code snippets in real-time. Users can choose to accept or reject a suggestion. When a suggestion is deemed useful (which is often the case) and accepted, users experience a significant increase in productivity. All platforms use a system of tab, enter, and arrow keys to choose among the options regarding the generated code, making it intuitive for any programmer to use.

Code generators such as Replit AI and the newly updated Copilot also provide a chat interface as an additional feature, enabling users to prompt the AI through either chats or block comments in the editor. ChatGPT-3.5 relies solely on the chat interface, and the generated code must be manually copied into a code editor.

The speed of the code generators was commendable, with most of them producing results within a few seconds. This is crucial for a code generator integrated into a code editor. The mode in which the code was generated was also noteworthy. In early tests, Llama and StarCoder appeared to produce code in “chunks” (1-5 lines at a time), while the other code generators would produce all the answers at once in a large block. In real coding scenarios, the mode in which the editors produce code depends on the amount of context present. With more initial context (code, block or line comments, etc.), code generators are capable of generating more code.

5 Discussion

5.1 The Impact of AI-Powered Code Generators on CS Education

The advent of AI-powered code generators poses significant challenges to computer science education. Studies have demonstrated the efficacy of these language models (LLMs) in solving programming problems commonly encountered in introductory computer science courses [3, 14]. This situation raises concerns for various stakeholders, including instructors, students, and administrators. As discussed in previous literature [14, 4], the use of AI-powered code generators raises serious concerns about their impact on the learning process. Students may opt to utilize these tools to complete assignments, potentially bypassing the intended learning experience. This raises concerns about the integrity of computer science education, as it can disadvantage students who do not use these tools and create an uneven playing field. Furthermore, there is a risk that students may develop a dependency on these tools, which could hinder their ability to develop independent problem-solving and coding skills. Educators must be cognizant of these challenges and develop strategies to mitigate their negative impact on the learning process.

5.2 Strategies for Addressing the Challenges

Several strategies can be employed to address the challenges posed by AI-powered code generators [7]. Each strategy presents its own advantages and disadvantages. One approach is to prohibit the use of LLMs in completing assignments. However, this may not be feasible or practical given the widespread availability of these tools. Additionally, enforcing such a ban would be challenging unless assignments are conducted as in-class activities under proctored conditions. This approach may only be viable for introductory computer science courses where assignments are relatively simple and can be completed within a single class session. However, even in this context, not all assignments may be suitable for completion within a single class session, depending on their complexity and the abilities of individual students.

An alternative approach is to modify assignments to make them less susceptible to automated solutions. This could involve creating more open-ended problems that require students to demonstrate their understanding of programming concepts and problem-solving skills. However, these open-ended problems may require more manual grading, making them less scalable for larger classes.

Another strategy is to embrace the use of LLMs as a learning tool and teach students how to utilize them effectively. Educators can provide guidance on the ethical use of these tools and encourage students to leverage them as resources to enhance their learning experience. This approach can help students develop their problem-solving and coding skills while benefiting from the assistance of AI-powered coding assistants.

5.3 Opportunities for Computer Science Education

The availability of AI-powered coding assistants also presents opportunities for educators to enhance the learning experience for students [4, 7, 14]. By integrating LLMs into the curriculum, educators can provide students with additional support and resources to help them learn programming concepts more effectively. LLMs can assist students in understanding complex topics, debugging code, and improving their coding skills. By leveraging these tools, educators can create a more engaging and interactive learning environment that encourages students to explore new concepts and develop their problem-solving skills.

Students enrolled in introductory programming courses often encounter challenges in devising effective coding strategies. When faced with debugging an error, students may find themselves at an impasse without immediate assistance. Traditional static learning resources, such as tutorials and textbooks, often fall short in providing immediate, contextualized, and interactive support. While teachers and teaching assistants (TAs) play a crucial role, their availability may be limited, and existing automated hint generation systems have inherent limitations. LLMs have the potential to address these issues by offering round-the-clock, contextualized support [1], thereby overcoming the limitations of traditional learning and tutoring methods.

6 Conclusion

In conclusion, our study assesses the performance of AI-powered code generators in solving introductory programming problems commonly used in computer science education. The findings reveal that these tools can be effective, with at least one generator solving each problem with 100% accuracy. While this presents challenges, such as students using AI-powered code generators bypassing the learning process and concerns about academic integrity, we believe that banning their use is not a viable solution. Instead, we propose incorporating these tools into the curriculum and teaching students how to use them effectively and ethically.

AI-powered code generators offer opportunities to enhance the learning experience by assisting students with complex topics, debugging, and improving their coding skills. Future work should focus on developing best practices, evaluating their impact on student outcomes, and addressing the challenges they pose. By leveraging these tools effectively, we can transform computer science education and empower students to succeed in the age of AI.

Acknowledgments

The authors would like to acknowledge the anonymous reviewers of the CCSC Southeast Conference for their invaluable feedback and constructive suggestions, which helped enhance the quality of this paper.

References

[1] Codio. *Codio Coach*. `https://www.codio.com/features/coach-ai-learning-assistant`. Accessed: 2024-06-05.

[2] Codio. *Student Perspectives and Use of Generative AI*. `https://www.codio.com/research/student-perspectives-and-use-of-gen-ai`. Accessed: 2024-06-05.

[3] Paul Denny, Viraj Kumar, and Nasser Giacaman. "Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 2023, pp. 1136–1142.

[4] Paul Denny et al. "Computing education in the era of generative AI". In: *Communications of the ACM* 67.2 (2024), pp. 56–67.

[5] GitHub. *GitHub Copilot · Your AI pair programmer*. `https://github.com/features/copilot`. Accessed: 2024-06-05.

[6] Shreya Johri. *The Making of ChatGPT: From Data to Dialogue*. `https://sitn.hms.harvard.edu/flash/2023/the-making-of-chatgpt-from-data-to-dialogue/`. Accessed: 2024-06-05.

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

[8] Meta. *Meta Llama*. https://llama.meta.com/. Accessed: 2024-06-05.

[9] OpenAI. *Introducing ChatGPT*. https://openai.com/index/chatgpt/. Accessed: 2024-06-05.

[10] Replit. *AI – Replit: Turn natural language into code*. https://replit.com/ai. Accessed: 2024-07-05.

[11] Amazon Web Services. *AI Code Generator – Amazon CodeWhisperer – AWS*. https://aws.amazon.com/codewhisperer/. Accessed: 2024-06-05.

[12] Duke University. *ComSci 101, Fall 2022: APTs*. https://courses.cs.duke.edu/fall22/compsci101/apt.php. Accessed: 2024-06-05.

[13] Duke University. *ComSci 201, Spring 2023: APTs*. https://courses.cs.duke.edu/spring23/compsci201/apt/secure/. Accessed: 2024-06-05.

[14] Tianjia Wang et al. "Exploring the Role of AI Assistants in Computer Science Education: Methods, Implications, and Instructor Perspectives". In: *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2023, pp. 92–102.

[15] Leandro von Werra and Loubna Ben Allal. *StarCoder: A State-of-the-Art LLM for Code*. https://huggingface.co/blog/starcoder. Accessed: 2024-07-05.

A Practical Approach to Improve CS Curriculum Through Continuous Assessment to ABET Outcomes[*]

Yi Liu and Gongbing Hong
Department of Information Systems and Computer Science
Georgia College and State University
Milledgeville, GA 31061
{yi.liu,gongbing.hong}@gcsu.edu

Abstract

This paper introduces a practical methodology to improve our CS curriculum using the continuous assessments conducted over the past decade. This methodology not only ensures effective improvements to curriculum, but also aims to reduce faculty workload associated with assessments. The approach includes integrating ABET-enabled student outcomes and the online assessment system from the university, optimizing the frequency of assessments by rotating a subset of student learning outcomes, focusing on the important components of our curriculum, establishing a routine meeting at the beginning of each fall semester aligned with the university's schedule, and consistently applying the collected results to restructure and refine the CS undergraduate curriculum at our college. As a result, the prerequisites of CS courses have been clarified, and the connections among CS courses have been strengthened. These curriculum improvements also help to enhance the quality of our CS program, as shown by ABET re-accreditation and higher rank from U.S. News and World Report.

[*]Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Given the rapidly evolving nature of computing, the CS curriculum—including the courses, course subjects, and course interconnections—must be continually updated in order to ensure that students are well-prepared upon graduation [4, 10]. However, despite the necessity and importance of regularly revising the curriculum for student success, there have been some notable challenges that arise.

First, any proposed changes must take into account various factors including expectations, program size, faculty resources, course schedules, and so on [11, 12]. For example, if a CS program faces budget constraints that may limit some upper-level CS courses to being offered only once a year, requiring such a course as a prerequisite could significantly negatively impact graduation rates. It could delay graduation for students who are under-performing, transferring, or changing their major to CS during their college years.

Second, curriculum changes can unintentionally introduce some unforeseen issues, causing confusion or new problems. One example is our modification of the course, Software Engineering(SE), which only required Data Structures as a prerequisite. But in order to align with the university's mission and the goal of refining the Knowledge-Skill-Disposition (K-S-D) framework [3, 7], we made this course as a capstone class, asking students to develop real-world projects with industrial clients. However, most projects required extensive database design and web programming. With Data Structures as the only prerequisite, many students lack the essential knowledge of database design, leading poor-quality or failed projects. Some even failed the course by the end.

Third, any curriculum changes require supports from faculty. Faculty must revise their courses by adding or removing related contents and spend time to implement these changes confidently. Oftentimes there is not enough time or opportunity for faculty to build the necessary collaboration and trust to support these changes [13]. Some studies [5, 10, 8] indicate that some resistances can occur due to perceived barriers to curriculum changes.

All the above challenges can be effectively addressed by using assessments. Assessment-data can help to overcome obstacles and boost confidence of faculty in the modified curriculum [6, 8, 13]. For example, after examining the data, our faculty discovered a higher than expected failure rate in the SE course. The data strongly supported revising the SE course prerequisites. Moreover, assessments collected after revising further verified that the change was necessary and effective.

However, faculty are usually busy with their teaching, research, and service responsibilities. Adding excessive assessment tasks can overwhelm them, leading to resistance to the assessment. To address this issue, it is important to design an assessment system that is both effective and manageable. The assessment process in our department aims at a simple implementation, and can easily adopted by the faculty without imposing unnecessary extra workloads for them.

In this paper, we share and summarize how ABET-guided assessments are used to effectively improve our curriculum. First, we introduce our Student Learning outcomes(SLOs) and the Performance Indicators(PIs) we have tailed from ABET. Second, we describe our structured assessment plan that aligns with ABET evaluation cycle. We provide some examples of the assessment data we have gathered. Then we discuss several examples of the major improvements made to the curriculum based on the assessment data, with our effort to minimize the workload of the faculty from the assessments. We conclude the paper by summarizing our key findings and their implications for ongoing curriculum development.

2 Methodology

2.1 Student Learning Outcomes Adopted from ABET

SLOs define the knowledge and skills that students are expected to possess by graduation. Our program, being ABET-accredited, benefited from directly adopting ABET's SLOs, which not only helped us on maintaining our accreditation status but also reduced the faculty's effort required when preparing the ABET re-accreditation, as this alignment simplified the assessment process. Our program's SLOs were tailored versions of the official ABET ones, as referenced in documents [2, 4]. In general, these outcomes were briefly reviewed in the CS assessment meeting. However, when ABET updated its criteria, we then allocated more time to update ours. Each SLO was associated with two PIs, which provided measurable metrics to evaluate the associated SLOs, ensuring an effective assessment process. Our SLOs are listed below.

SLO1: Apply computer science theory and software development fundamentals to produce computing-based solutions.

PI 1. Students will be able to demonstrate and apply knowledge of mathematical functions to analyze a given algorithm.

PI 2. Students will be able to recognize appropriate algorithm to solve a problem.

SLO2: Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.

PI 1. Students will be able to recognize design and development principles.

PI 2. Students will be able to implement and evaluate the designed solution for a given problem.

SLO3: Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.

PI 1. Students will be able to share in work of team .

PI 2. Students will be able to fulfill duties of team roles.

SLO4: Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.

PI 1. Students will be able to identify professional, legal, and ethical issues.

PI 2. Students will be able to understand professional, legal, and ethical responsibilities.

SLO5: Communicate effectively in a variety of professional contexts.

PI 1. Students will be able to produce a variety of documents for technical and nontechnical audiences.

PI 2. Students will be able to prepare and deliver oral presentations.

SLO6: Analyze a complex computing problem and apply principles of computing and other relevant disciplines to identify solutions.

PI 1. Students will be able to analyze a complex computing problem.

PI 2. Students will be able to apply computing principles to identify solutions.

2.2 Assessment Plan

After defining the SLOs and the corresponding PIs, we quickly laid out the curriculum maps from CS courses to SLOs. The maps showed the alignments between the SLOs and what were taught in our curriculum. One of the maps was specifically for assessment, and shown in Table 1. A detailed assessment plan was then developed based on the map to ensure the successful evaluation of learning. [2, 6]. The assessment plan included the following steps:

1) Select the subsets of SLOs to assess each year on a rotating basis.

2) Select courses from the curriculum map to evaluate for the upcoming academic year.

3) Choose course specific learning outcomes, means of assessment, desired standard of achievement based on PIs of the chosen SLO.

4) Collect the assessment data and perform analysis.

Table 1: CS SLOs Assessed by Degree-Required Courses

CS Courses	**CSCI 2800**	**CSCI 3211**	**CSCI 3212**	**CSCI 3341**	**CSCI 3342**	**CSCI 3343**	**CSCI 4320**	**CSCI 4330**	**CSCI 4520**	**CSCI 4710**
SLO1: Apply computer science theory and software development fundamentals to produce computing-based solutions.									X	
SLO2: Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.				X		X		X		X
SLO3: Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.			X		X		X		X	X
SLO4: Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.	X				X					
SLO5: Communicate effectively in a variety of professional contexts.								X		X
SLO6: Analyze a complex computing problem and apply principles of computing and other relevant disciplines to identify solutions.		X	X			X	X	X	X	X

5) Discuss the assessment results at the beginning of the next fall semester and propose necessary improvements to curriculum, faculty resources, schedules, and other related areas.

To manage the faculty's assessment workload effectively, we only focused on degree-required upper-level courses. Lower-level courses and elective courses were excluded from the assessment scope in general. The only lower-level course to be assessed was CSCI2800, which focused on social and professional issues in computing. This approach enabled us to target our assessment efforts on the critical elements of our curriculum.

Table 1 showed our curriculum map used for assessments. Typically, direct assessments were carried out by collecting data from our major-required upper-level courses. This map specified which courses were used to evaluate each of SLOs. The detailed assessment plan was defined accordingly and revised as needed. Our initial plan was defined in 2010 and was revised in 2015. In general, only half of the SLOs were assessed each year.

2.3 Assessment Data Collection

The process of the curriculum improvement needs effectively to use assessment data. The quantitative data from assessments measure the students' knowledge and skills being taught in CS courses. In our department, the assessment data were collected using the form as shown in Table 2. The form was also used across the college and the university. By using the same form as the university, we were able to reduce faculty workload when integrating assessments with the university's online assessment system. At the beginning of each fall semester, faculty members selected courses to assess a rotating subset of SLOs. One faculty member usually assessed one course per year. The faculty could define their own metrics to assess with the flexibility to choose the course components, the measurement methods, and the desired outcomes. Since this plan spanned multiple years, a metric could be as simple as a quiz question, an assignment, a project, or one exam component. There were no specific requirements regarding the number of questions or assignments for assessment. Table 2 provided an example of the assessment data we collected.

Table 2: Example of the Assessments to PI1 of SLO2 Using the Course Software Engineering

SLO2	Design, implement, and evaluate a computing based solution to meet a given set of computing requirements.
PI1	Students will be able to recognize design and development principles.
Measure	Final Exam Question -: Provide a description of the Design Patterns that were mentioned in the required videos.
Details/Description	Describe how you were able to use them (or could have used them) in code that you have completed in any of the required courses for CS majors.
Desired Standard of Achievement	80% of students should answer the questions correctly.
Summary of Findings	Only 60% students successfully described usage of design patterns in current or former projects on the final exam.
Was this outcome met?	No

Explain your results	Students needed to be able to demonstrate understanding of design patterns and recognize where they have been use in former projects. The results did not turn out as expected because of the length of the test. By the time that students got to this question, there was little attention paid to making sure the question was completely answered
Action	more work is needed

2.4 Improving CS Curriculum Using Assessment Data

Table 3: Example of Major Improvements to CS Curriculum

Meeting Time	10/21/16
Improvement	Add a new course CSCI 3343 Computer Systems Security
Details	The assessment data showed that using several courses to cover the different topics of security was not satisfactory. The connections were missing among some security technologies. Some core security topics were covered without sufficient depth. The new course computer system security should be offered.

If the curriculum needs to change, the assessment data help to reduce resistance to the changes, and provide confidence when faculty doubt about whether such change is worth or not [4, 7]. At the beginning of each fall semester, our annual assessment meeting was held where faculty reviewed data, discussed issues related to the assessment, and shared their insights. If needed, changes to courses or the curriculum were proposed during these discussions. The meeting provided a platform to promote collaboration, enhance confidence among faculty, and support improvements to the curriculum. The discussions of many related issues in the meeting ensured that the changes to the curriculum were implemented correctly. Any changes were then assessed in subsequent assessment cycles. These assessment cycles provided continual feedback on the improvements and helped faculty decide if the changes were beneficial.

3 Discussions of Effectiveness of the Methodology

3.1 Major Improvements of the Curriculum

Continuous improvements across multiple years are essential to help CS graduates remain competitive and relevant. With the methodology described above, our program made many improvements. Some of the major improvements were listed in Table 4. The table listed the dates these changes were made and the reasons for them. The major improvements in our paper were defined as the ones which usually needed to approve by the college and the university. Examples include altering prerequisites, adding, deleting, and replacing courses.

Changing prerequisites of a course typically occurs after substantial updates to course components, aiming to align with current educational outcomes and maintain relevance in this rapidly evolving field. But there are challenges related to changing prerequisites. One is whether it is necessary to use an upper-level CS course as a prerequisite. This change could delay graduation for students who are under-performing, transferring, or changing majors. Potentially, it may lead to a negative impact on the overall graduation rate. It may also create unnecessary barriers for students intending to switch into CS, especially in the programs where some major-required upper-level courses are offered only once per year.

Our approach to address the above challenge was to assess the course before and after the change, and then compare the assessment data to ensure that the change was necessary and beneficial. For example, when the course, SE, was chosen to be a capstone class, one of the course components was to require students to engage with real-world projects, which often needed to design and implement a database system. After several years of assessments, substituting CSCI3410 Data Structures with CSCI4710 Database Systems as the prerequisite was proven to be very beneficial. This modification was documented in the first row of Table 4. Similar changes involving new prerequisites or co-requisites were recorded in rows 3, 4, 7, and 8 of the table.

Another challenge is about whether to add a course as required. Adding more major-required courses can be as complicate as using upper-level CS courses as prerequisite, potentially increasing students' educational costs and time commitments, and reducing flexibility to choose electives. For example, the course on Computer Security was introduced due to increasing vulnerabilities to cyber threats for companies, and a requirement emphasized by ABET. There was a high demand if students can be proficient in computer security too. Originally, several courses in our program covered aspects of security, but assessments revealed that there were some gaps in topic interrelations, resulting in insufficient breadth and depth. Collecting all assessment data for the topic during the ABET review were also time-consuming and sometimes confusing. So the course, Computer System Security was proposed in 2016 and became a required one in 2017. Meanwhile, the course, Parallel & Distributed Computing, remained an elective. These changes were recorded in Table 4. There was no major improvements in 2020, 2021, and 2022 due to the disruptions caused by the pandemic. Taking actions could be challenging if the assessment results were not met, as classroom settings and teaching methods had changed. No major improvement was proposed in 2023 either. The minor improvements including updating course components to assess SLOs, adjusting course schedules, and so on, were not mentioned in the paper due to page limits, but can then be found on our web site [1].

Table 4: Examples of Major Improvements to CS Curriculum

Date	**Change**	**Reason**
10/6/2015	Changed the prerequisite of CSCI4320 Software Engineering from CSCI3410 Intro to Data Structures to CSCI4710 Databases	The students who did not take the course related to database usually did not do well on the real-world course projects.
10/6/2015	Reactivated the following courses: CSCI2800 Social & Professional Issues	Several CS courses provided professional, ethical, legal, security, and global issues and responsibilities in computer science related fields. However, fulfilling the ABET requirement related to SLO4 was time-consuming, and insufficient. The simple solution was to reactivate the course CSCI2800.
11/3/2015	Changed prerequisite of CSCI3342 System & Network Programming from CSCI3410, to CSCI3341.	The assessment results showed that it was difficult to address advanced topics without the foundations of OS.

Continued on next page

Date	Change	Reason
11/3/2015	Added CSCI3341 Operating System as a prerequisite to CSCI4950 Special Topics	CSCI4950 was an option for the CS capstone course and should require knowledge of the upper-level CS courses. Allowing students to take a "capstone" course without knowledge of Advanced CS courses was an issue raised especially during the ABET review.
10/21/16	Add a new course CSCI3343 Computer Systems Security	The assessment data showed that using several courses to cover the different topics of security was not satisfactory. The connections were missing among some security technologies. Some core security topics were covered without sufficient depth. The new course for security should be offered.
1/20/17	Changed the prerequisite for CSCI3342 Systems and Networking Programming (Added CSCI2350 Programming II)	To complete the programming assignments, students needed knowledge of C/C++ provided in CSCI2350.
1/20/2017	The co-requisite CSCI2350 Programming II was added for the course CSCI3211 Assembly Language.	Upon completing an assessment cycle, the instructor of CSCI3211 concluded that some knowledge of C/C++ programming was necessary.
9/8/2017	CSCI3343 computer systems security became one of the major required courses.	The course resolved the problems encountered in the previous assessment cycle. Given the importance of cyber security, the course was made a major-required course.

Continued on next page

Date	Change	Reason
9/8/2017	CSCI4800 Parallel & Distributed Comp was continued to be offered as an elective.	Topics in parallel and distributed computing were adequately covered in the current required courses, including network programming, operating systems, and Algorithms.
5/10/2019	CSCI3212, CSCI3341, CSCI4520 provided sufficient exposure to parallel and distributed computing topics.	The assessment data further indicated that the goal of exposing students to parallel and distributed computing topics was achieved through several existing courses. Therefore, it was unnecessary to make parallel computing as required.

3.2 Ease Faculty Workload When Developing Practical Assessment Plan

Considering that an assessment plan usually extends over multiple years, it is important to minimize unnecessary faculty workload during this time. Our strategy involved simplifying the complexity of assessment tasks by implementing the following aspects.

First, we adopted ABET's SLOs and made minor adjustments to align them with our program and the university's mission. We updated our SLOs only when ABET updated theirs and enforced the new requirements.

Second, we optimized the frequency of assessments to each SLO, and focused on the important components of curriculum. To do so, We assessed half of the SLOs each year, allowing each SLO to be assessed three times before the next ABET review. Generally, a faculty member assessed only one course per year. This method enabled us to complete at least one assessment cycle, and implemented some improvements when the outcomes didn't meet expectations. In terms of selecting courses for assessment, we focused on major-required upper-level courses. Since these courses integrated various critical aspects of CS field, assessing them could quickly identify strength and weakness of students' knowledge and skills. Therefore, it further simplified the assessment process and reduced the faculty's time needed for assessments.

Third, faculty members were given the flexibility to select the course, course components, and methods to assess the SLOs. They even could choose a single assignment or an exam/quiz question for the assessment as long as they considered it as appropriate. In general, each faculty member only needed to complete a form similar to table 2 each year.

Fourth, we adopted the university's online assessment system for data management. By using the same online assessment system with the university, faculty assessment workloads could be further reduced by eliminating the need for duplicate data entry and additional training.

Fifth, our annual assessment meeting was limited to 1-2 hours. In the initial years of implementing the assessment plan, we encountered more curriculum issues than expected, resulting in a longer meeting. However, after 2017, fewer curriculum changes were needed, and consequently, the length of the meeting was significantly reduced. A one-hour meeting per year was sufficient to discuss the collected assessment data, update the curriculum, and plan the assessment for the next academic year.

Generally, each faculty member was expected to spend approximately 6-8 hours annually for assessment. However, the assessment coordinator needed additional time to generate a summary report and ensured that the assessment data were collected completely and consistently.

4 Conclusion

A practical methodology to improve our curriculum based on a decade of assessments is introduced in this paper. Establishing a multi-year assessment strategy is essential for continuous curriculum improvement. Meanwhile, implementing an assessment plan that minimizes faculty resistance is equally important. This paper first introduces our assessment plan, addresses several challenges in curriculum development through these assessments, and demonstrates the improvements to our curriculum. It also shares strategies to lighten the assessment workload for our faculty. After several years of conducting assessments, we conclude that our approach not only provides consistent guidance for improving the curriculum, but also simplifies the assessment process for faculty. As a result, the course prerequisites have been clarified, and the inter-course collaborations are enhanced. Moreover, these improvements have helped us to obtain and maintain the ABET accreditation, and to move our ranking to No.#1 among public regional universities in the state of Georgia at the year of 2024 by US News & World Report [9].

References

[1] ABET. *ABET Self-Study Report for the Bachelor of Science Computer Science.* (be added after blinded review). 2021.

[2] ABET. *ABET-Assessment Planning.* `https://www.abet.org/accreditation/get-accredited/assessment-planning/`.

[3] ABET. *Criteria for Accrediting Computing Programs, 2017 – 2018.* `https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2017-2018/`.

[4] ACM. *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`.

[5] Charlotte L. Briggs. "Curriculum collaborations: A key to continuous program renewal". In: *The Journal of Higher Education* 7.6 (2007), pp. 679–711.

[6] Loretta Donovan, Tim D. Green, and Candice Mason. "Examining the 21st century classroom: Developing an innovation configuration map". In: *Journal of Educational Computing Research* 50.2 (2014), pp. 161–178.

[7] Stephen Frezza et al. "Modelling competencies for computing education beyond 2020: A research-based approach to defining competencies in the computing disciplines". In: *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education.* 2018, pp. 148–174.

[8] Gene E. Hall and Shirley M. Hord. *Implementing change: Patterns, principles, and potholes.* 4th. Upper Saddle River, NJ: Pearson, 2015.

[9] US News & World Report. *Rank by 2024 US News & World Report.* `https://www.gcsu.edu/about/recognitions`.

[10] Mehran Sahami and et al. "ACM/IEEE-CS computer science curriculum 2013: reviewing the ironman report". In: *Proceedings of the 44th ACM Technical Symposium on Computer Science Education.* 2013.

[11] Shingo Takada, Ernesto Cuadros-Vargas, and et al. "Toward the visual understanding of computing curricula". In: *Education and Information Technologies* 25 (2020), pp. 4231–4270.

[12] Leslie J. Waguespack. "Adopting Competency Mindful of Professionalism in Baccalaureate Computing Curricula". In: *EDSIGCON.* 2019.

[13] Jon W. Wiles and Joseph C. Bondi. *Curriculum development: A guide to practice.* 9th. Boston, MA: Pearson, 2014.

A Machine Learning Based Sentiment Analysis for Twitter Data*

Kazi Abdullah Al Arafat[1], Kode Creer[2],
Mahmudur Rahman Roni[1], Imtiaz Parvez[2]
[1]Dept. of Computer Science and Engineering
Atish Dipankar University of Science and Technology
Dhaka, Bangladesh
{cse2010004,mahmud.roni}@adust.edu.bd
[2]Dept. of Computer Science
Utah Valley University
Orem, UT 84058
{kode.creer,imtiaz.parvez}@uvu.edu

Abstract

Sentiment analysis, also known as opinion mining, is a computational study of people's opinions, sentiments, evaluations, attitudes, and emotions expressed in textual data. This study explores the application of machine learning algorithms for sentiment analysis on a preprocessed dataset. The study employs Support Vector Machines (SVM), Maximum Entropy (Max Ent), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN) for sentiment classification. The process includes feature extraction, model training, and evaluation using standard classification metrics. Out of the four algorithms, SVM, CNN, and RNN scored 97% accuracy, while Max Ent achieved a slightly lower 95% accuracy. More specifically, in sentiment analysis, SVM demonstrated overall better performances.

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Sentiment analysis, a branch of natural language processing (NLP), is essential for interpreting beliefs, attitudes, and feelings that are present in textual information. Opinion analysis has become increasingly important in today's digitally driven world, as it offers a vital tool for understanding public opinion, interpreting consumer feedback, and monitoring social media trends.

Sentiment analysis can interpret feelings in textual data, which has become essential for a variety of applications, including the predictive understanding of market trends, the strategic management of brand reputation, and the detailed analysis of consumer feedback. Machine learning techniques show promise as solutions since they can improve and automate sentiment analysis. Businesses and scholars can gain valuable insights from the large amount of textual data available in the digital sphere by utilizing these tools.

In this modern era of rapid communication and exchange of information, the analysis of sentiments holds great importance for decision-makers in different fields. It acts as a crucial instrument in assisting them to gauge the emotions and attitudes exhibited by the general public. Moreover, sentiment analysis aids in determining the most appropriate funding agency for a specific situation or project entities to negotiate the complex terrain of human expression in the digital era, whether it be for predicting market dynamics, measuring social responses to unfolding events, or knowing the pulse of consumers. The goal of this research is to evaluate the efficacy of traditional machine learning and state-of-the-art neural network models for sentiment analysis in classifying textual input into positive, negative, and neutral feelings. Situated at the real-world application crossroads, where technological innovation thrives.

Many researchers have worked extensively in the field of sentiment analysis in recent years, employing a variety of methodologies. To polarise the opinions, binary classification techniques were initially employed. Researchers use a variety of classification techniques to categorize the sentiments because of the diversity of data available nowadays. In [3, 8, 10, 13], the authors explore diverse machine learning methods on sentiment analysis in Twitter data. They contrasted different machine-learning techniques to classify tweets as neutral, negative, or positive. Tokenization and removing non-English words are the preprocessing steps for the data in [10]. The machine learning model is then fed processed data in order to classify the emotions. Of the classifiers used in this article, the Naive Bayes Classifier has an accuracy of 86%. An algorithm to categorize the emotions tweeted in daily life has been proposed in [13]. They first examine the sentiment polarity of every tweet in the dataset of [3]. Then, the tweet is classified as positive if the sentiment polarity is greater than zero, and as neutral if it is equal to zero. The tweet is negative otherwise. The real-time data set gathered by the Twitter API was the subject of all three of these articles.

The study [18] includes a brief review of the following topics: data collection, preprocessing, different sentiment analysis approaches, feature extraction techniques, assessment metrics, real-world applications, and challenges in sentiment analysis. In [4, 16], the authors use the natural language processing toolkit (NLTK) to illustrate the sentiment analysis for a specific product that was posted on Twitter. Bag of Words (BoW) and Term Frequency-inverse (TF-IDF) have been integrated to analyse sentiments as neutral, negative, or positive. They discovered that the accuracy of sentiment analysis increased when they took advantage of the TF-IDF vectorizer. According to simulation results, the suggested system's efficiency was 85.25%.

In [5, 14], authors gathered information on movie reviews and Twitter API from IMDB. To categorize the sentiments, they used Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Naive Bayes (NB). When compared to the other two classifiers, ANN performed the best in terms of accuracy in both datasets. In [1], a deep recurrent network with long short term memory (LSTM) is used to classify various emotions within the bangla text. This method's accuracy in categorising emotions was 94%.

In this study, we scrutinized a contemporary dataset using four distinct machine learning algorithms, resulting in the attainment of remarkable accuracy levels. Our exploration contributes valuable insights into the efficacy of these algorithms for sentiment analysis on current datasets, underscoring their potential in real-world applications.

The paper is organized as follows: Section 2 discusses the methodology, Section 3 presents the simulation and results analysis, and Section 4 includes concluding remarks and future work.

2 Methodology

In this segment, we will put forth our suggested technique for scrutinizing emotions expressed on Twitter. To provide visual clarity to our approach, Figure 1 showcases the framework that serves as our guide. To handle the information collected for this assignment, we made use of the natural language processing toolkit (NLTK). Following the processing of the data, we employed a machine learning classifier to classify it.

2.1 Data Collection

We obtained a raw dataset from Kaggle to conduct sentiment analysis. The data collection process occurred in April and May of 2023, deliberately chosen to encompass a wide array of tweets that could potentially reveal changing

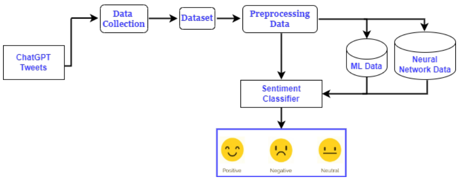


Figure 1: Sentiment Analysis Process Framework

trends in sentiment. Throughout this time frame, data gathering was conducted consistently, resulting in a varied collection of tweets. At present, the dataset comprises 66,375 tweets that are prepared for further examination and analysis. From the total data set, 75% is used for training while the rest data is used for testing. A statistic of data utilization is shown in Table 1.

Table 1: Statistics of Dataset

Dataset	Positive	Negative	Neutral	Total
Training	8620	3254	41226	53100
Test	2164	821	10290	13275

2.2 Data Preprocessing

Sometimes it can be difficult to extract keywords from a dataset due to the presence of several factors, including stop words, emojis, junk data, misspellings, stop words, hashtags, special characters, and more. A successful data analysis may be significantly hampered by these factors. Hashtags, for instance, are frequently used on social media sites to aid in content discovery; however, they may make keyword extraction more difficult. But, by serving as linkers or clarifiers, special characters fill in the blanks where words alone might fail to fully express a phrase's meaning. Although they can be challenging to correctly interpret, emojis give words a more sentimental feel.

In order to get around these obstacles, a thorough pretreatment process is carried out before using keyword extraction methods. The data cleaning process involves removing any duplicate tweets, irrelevant information, and spammy content that might have been unintentionally collected. Also, efforts were made to standardise text encoding and resolve character encoding problems, which are prevalent in data sourced from social media platforms. A crucial step in text processing was achieved by extracting partially separated tokens (words or subwords) from the tweets. The particularities of social media conversation, like the use of emojis, hashtags, and mentions, were taken into account during the tokenization process. Distracting text elements like acronyms and uncommon abbreviations that are commonly used on Twitter were also given special attention, along with spelling mistakes. To further improve the quality of the dataset, stopwords—words that are frequently found in text but rarely have a significant impact on sentiment analysis—were also filtered out. The objective of preprocessing decisions was to remove superfluous noise while preserving the important content of tweets. The thorough pretreatment ensured that the sentiment analysis data was of the highest calibre and appropriate for further analysis and modelling.

2.3 Feature Extraction

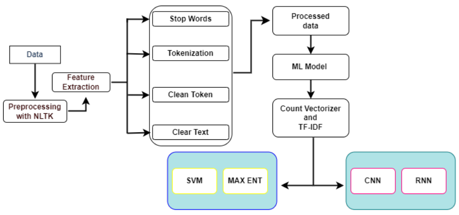


Figure 2: Data Processing and Feature Extraction Framework

As it entails transforming unprocessed textual data into a format appropriate for machine learning models, feature extraction is a crucial preprocessing stage in sentiment analysis. Two methods for feature extraction are used in this study:

- Count Vectorization: This technique uses a matrix of word counts to represent the text. Every document is converted into a vector, with every element denoting the number of that word in the document.
- TF-IDF Transformation: Word importance in a document is evaluated in relation to its frequency across the entire dataset using the term frequency-inverse document frequency (TF-IDF) method. It aids in emphasizing words that are unique to a given document.

By converting textual data into numerical representations, these methods seek to improve the efficiency with which machine learning models process and recognize patterns.

2.4 Sentiment Analysis Algorithms

Support Vector Machines (SVM) Support Vector Machines (SVM) excel in classification tasks by finding the optimal hyperplane that separates data points into different classes [17, 15]. In the realm of sentiment analysis, SVM is a powerful tool for discerning the sentiment polarity of a given text, classifying it as positive, negative, or neutral. SVM's strength lies in its ability to handle high-dimensional data efficiently and find clear decision boundaries, making it well-suited for the nuanced task of sentiment classification.

Maximum Entropy (Max Ent) MaxEnt, frequently implemented using Logistic Regression, stands as a probabilistic model gauging the likelihood of a sample belonging to a particular class [2, 6]. This method proves itself valuable in sentiment analysis, foreseeing the probability of a text being categorized into positive, negative, or neutral sentiment classes. While embracing simplicity, Logistic Regression demonstrates effectiveness in capturing intricate relationships within textual data. Its capability offers insights into nuanced sentiments, making a meaningful contribution to the broader sentiment analysis field.

Convolutional Neural Networks (CNN) Originally designed for image processing, Convolutional Neural Networks (CNNs) have demonstrated remarkable efficacy in processing sequential data, including text [9, 7]. In sentiment analysis, CNNs leverage convolutional layers to capture local patterns and hierarchical representations of textual features. Their ability to discern intricate details and relationships between words makes CNNs a powerful choice for sentiment classification tasks, allowing them to effectively capture the nuanced sentiment expressed in natural language text.

Recurrent Neural Networks (RNN) Recurrent Neural Networks (RNNs), specifically Bidirectional Long Short-Term Memory (LSTM) networks, are designed to handle sequential data by maintaining memory of previous inputs [11, 12]. In sentiment analysis, RNNs excel at capturing contextual dependencies between words in a sentence. The Bidirectional LSTM enhances this capability by processing the input sequence in both forward and backward directions, allowing the network to capture dependencies from both past and future contexts. Despite facing challenges like vanishing gradients, RNNs, including Bidirectional LSTMs, remain valuable tools in sentiment analysis, providing a nuanced understanding of sentiment-related information in natural language text.

2.5 Model Training and Evaluation

A crucial stage in the process of developing sentiment analysis models is the training phase, which entails using labeled datasets to help each model learn to identify patterns linked to neutral, positive, or negative sentiments. This training process is carried out on a variety of models in the context of the provided code, such as Support Vector Machines (SVM), Maximum Entropy (Max Ent), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN) with Bidirectional LSTMs. In the process of training, the models learn from the features that are extracted from the textual data to optimise their parameters, which enables them to identify the subtleties of sentiment found in the training dataset.

In order to achieve accurate sentiment classification, model parameters must be optimized. This allows the models to adjust to the unique features of the dataset. Improving the model's capacity to effectively generalise to untested data is the goal in order to guarantee reliable sentiment analysis for a variety of textual inputs. Weights and biases are adjusted during the training process, especially for neural network models, in order to reduce the discrepancy between the true and predicted sentiments in the training set.

Every sentiment analysis model is tested on the same dataset set aside for evaluation after the training phase is over. In order to evaluate how well the models generalize to new, unobserved examples, this test dataset acts as a benchmark. Evaluation metrics are used to quantify each model's performance in sentiment classification, including accuracy, precision, recall, and F1-score. These metrics offer a thorough evaluation of the model's efficacy in precisely predicting positive, negative, and neutral sentiments in real-world textual data, as well as insightful information about the model's advantages and disadvantages.

3 Simulation and Result

The precision metric in sentiment analysis is crucial for understanding the reliability of positive, negative, and neutral predictions made by each model. It measures the ratio of correctly predicted instances of a sentiment class to the total instances predicted for that class. Here's a detailed discussion of precision scores for each sentiment class and their implications:

Support Vector Machines (SVM) SVM achieves high precision for neutral sentiments (0.98), indicating its proficiency in correctly classifying neutral texts. However, the precision for negative sentiments is slightly lower (0.94), suggesting a small proportion of false positives. The comprehensive findings are encapsulated in Figure 3 and Table 2, where they respectively provide visual and tabular representations summarizing key metrics and outcomes.

Maximum Entropy (Max Ent) Logistic Regression, representing Maximum Entropy, demonstrates strong precision for positive (0.89) and neutral

Table 2: SVM Classification Performance

Categories	Precision	Recall	f1-score	Support
Negative	0.94	0.76	0.84	821
Neutral	0.98	0.99	0.98	10290
Positive	0.92	0.93	0.92	2164
Accuracy			0.97	13275
Macro avg	0.95	0.89	0.92	13275
Weighted avg	0.96	0.97	0.96	13275

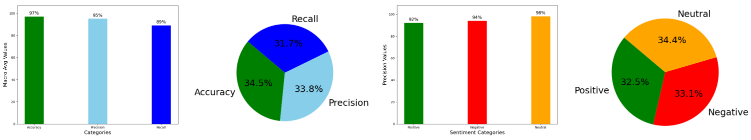


Figure 3: (a) SVM Classification Accuracy and Macro Avg Values Bar Graph (b) SVM Classification Accuracy and Macro Avg Values Pie Chart (c) SVM Sentiment Precision Values Bar Graph (d) SVM Sentiment Precision Values Pie Chart

(0.96) sentiments. However, it exhibits higher precision for negative sentiments (0.96), implying some false positive predictions for negativity. The comprehensive findings are encapsulated in Figure 4 and Table 3, where they respectively provide visual and tabular representations summarizing key metrics and outcomes.

Table 3: Max Ent Classification Performance

Categories	Precision	Recall	f1-score	Support
Negative	0.96	0.63	0.76	821
Neutral	0.96	0.99	0.97	10290
Positive	0.89	0.89	0.89	2164
Accuracy			0.95	13275
Macro avg	0.94	0.83	0.87	13275
Weighted avg	0.95	0.95	0.95	13275

Convolutional Neural Networks (CNN) CNN consistently displays robust precision across all sentiment classes, with notable precision values for

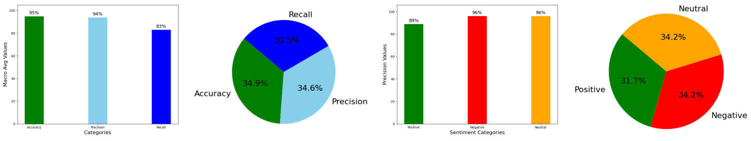


Figure 4: (a) Max Ent Classification Accuracy and Macro Avg Values Bar Graph (b) Max Ent Classification Accuracy and Macro Avg Values Pie Chart (c) Max Ent Sentiment Precision Values Bar Graph (d) Max Ent Sentiment Precision Values Pie Chart

positive (0.93) and negative (0.86) sentiments. This indicates its effectiveness in avoiding false positives for both polarities. The comprehensive findings are encapsulated in Figure 5 and Table 4, where they respectively provide visual and tabular representations summarizing key metrics and outcomes.

Table 4: CNN Classification Performance

Categories	Precision	Recall	f1-score	Support
Negative	0.86	0.82	0.84	821
Neutral	0.98	0.99	0.99	10290
Positive	0.93	0.94	0.94	2164
Accuracy			0.97	13275
Macro avg	0.93	0.91	0.92	13275
Weighted avg	0.97	0.97	0.97	13275

Recurrent Neural Networks (RNN) Bidirectional LSTMs in the RNN model exhibit high precision for neutral (0.98) and positive (0.93) sentiments. However, the precision for negative sentiments is comparatively lower (0.88), suggesting some misclassification. The comprehensive findings are encapsulated in Figure 6 and Table 5, where they respectively provide visual and tabular representations summarizing key metrics and outcomes.

Comparative Analysis Positive Sentiments: CNN and RNN with Bidirectional LSTMs perform well, with precision scores of 0.93, respectively.

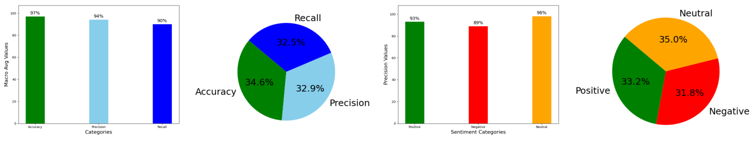


Figure 5: (a) CNN Classification Accuracy and Macro Avg Values Bar Graph (b) CNN Classification Accuracy and Macro Avg Values Pie Chart (c) CNN Sentiment Precision Values Bar Graph (d) CNN Sentiment Precision Values Pie Chart

Table 5: RNN Classification Performance

Categories	Precision	Recall	f1-score	Support
Negative	0.88	0.79	0.83	821
Neutral	0.98	0.99	0.99	10290
Positive	0.93	0.94	0.93	2164
Accuracy			0.97	13275
Macro avg	0.93	0.91	0.92	13275
Weighted avg	0.97	0.97	0.97	13275

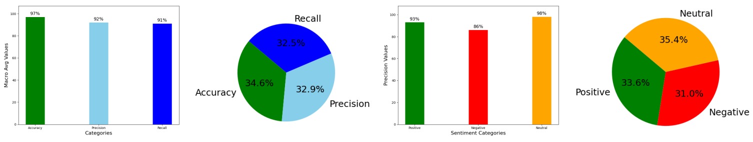


Figure 6: (a) RNN Classification Accuracy and Macro Avg Values Bar Graph (b) RNN Classification Accuracy and Macro Avg Values Pie Chart (c) RNN Sentiment Precision Values Bar Graph (d) RNN Sentiment Precision Values Pie Chart

Neutral Sentiments: RNN with Bidirectional LSTMs achieves the highest precision (0.98), equivalent to SVM (0.98).

Negative Sentiments: CNN and SVM exhibit precision scores 0.86 and 0.94, respectively, while Logistic Regression lags slightly behind 0.96. The comprehensive findings are encapsulated in Figure 7, where they respectively provide visual representations summarizing key metrics and outcomes.

Overall Assessment: While CNN and RNN with Bidirectional LSTMs showcase competitive precision values, the choice between them may depend on other factors like computational efficiency and interpretability.
SVM, despite slightly lower precision for negative sentiments, maintains a strong overall performance, especially in discerning neutral sentiments. Logistic Regression excels in precision for positive and neutral sentiments but may benefit from improvements in predicting negative sentiments.
The precision metric provides valuable insights into the strengths and weaknesses of each model in handling specific sentiment categories, aiding in the selection of the most suitable model based on task requirements and priorities.

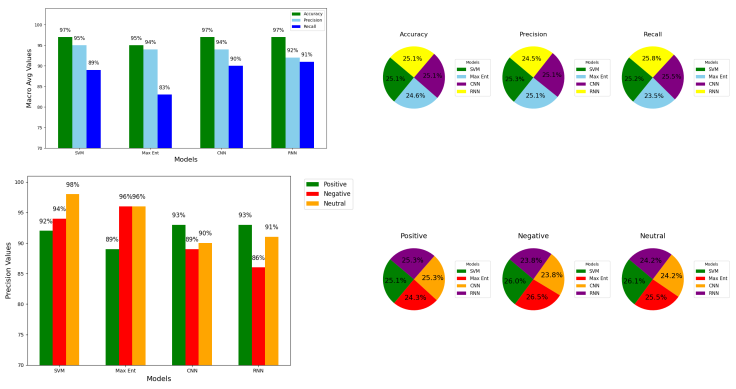


Figure 7: (a) Comparison Classification of SVM, Max Ent, CNN, RNN Bar Graph (b) Comparison Classification of SVM, Max Ent, CNN, RNN Pie Chart (c) Comparison Sentiment Categories of SVM, Max Ent, CNN, RNN Bar Graph (d) Comparison Sentiment Categories of SVM, Max Ent, CNN, RNN Pie Chart

4 Conclusion

This study explores various machine learning algorithms for sentiment analysis on textual Twitter data, revealing that SVM, CNN, and RNN have achieved remarkable accuracy of 97%, while Max Ent has scored slightly lower accuracy of 95%. Notably, SVM exhibited superior precision in sentiment classification, making it a robust choice. The findings guide model selection and lay a foundation for advancing sentiment analysis techniques. Additionally, this study can be used as classroom material for demonstrating the various machine learning algorithms and their comparative performance. Future work may involve developing an automated system, integrating these models for real-time analysis of diverse textual data across evolving language patterns.

5 Acknowledgement

To improve the readability and quality of language, all parts of this paper have been grammatically revised using Quillbot AI tool.

References

[1] Afrin Ahmed and Mohammad Abu Yousuf. "Sentiment analysis on Bangla text using long short-term memory (LSTM) recurrent neural network". In: *Proceedings of International Conference on Trends in Computational and Cognitive Engineering: Proceedings of TCCE 2020*. Springer. 2020, pp. 181–192.

[2] Owusu Fordjour Aidoo et al. "Model-based prediction of the potential geographical distribution of the invasive coconut mite, Aceria guerreronis Keifer (Acari: Eriophyidae) based on MaxEnt". In: *Agricultural and Forest Entomology* 24.3 (2022), pp. 390–404.

[3] Sanjeev Dhawan, Kulvinder Singh, and Priyanka Chauhan. "Sentiment analysis of Twitter data in online social network". In: *2019 5th International Conference on Signal Processing, Computing and Control (ISPCC)*. IEEE. 2019, pp. 255–259.

[4] Sahar A. El Rahman, Feddah Alhumaidi AlOtaibi, and Wejdan Abdullah AlShehri. "Sentiment Analysis of Twitter Data". In: *2019 International Conference on Computer and Information Sciences (ICCIS)*. 2019, pp. 1–4. DOI: 10.1109/ICCISci.2019.8716464.

[5] Ali Hasan et al. "Machine learning-based sentiment analysis for twitter accounts". In: *Mathematical and computational applications* 23.1 (2018), p. 11.

[6] Edwin T Jaynes. "On the rationale of maximum-entropy methods". In: *Proceedings of the IEEE* 70.9 (1982), pp. 939–952.

[7] Zhao Jianqiang, Gui Xiaolin, and Zhang Xuejun. “Deep convolution neural networks for twitter sentiment analysis”. In: *IEEE access* 6 (2018), pp. 23253–23260.

[8] Mantasha Khan and Ankita Srivastava. “Sentiment analysis of Twitter data using machine learning techniques”. In: *International Journal of Engineering and Management Research* 14.1 (2024), pp. 196–203.

[9] Zewen Li et al. “A survey of convolutional neural networks: analysis, applications, and prospects”. In: *IEEE transactions on neural networks and learning systems* (2021).

[10] Lokesh Mandloi and Ruchi Patel. “Twitter sentiments analysis using machine learninig methods”. In: *2020 international conference for emerging technology (INCET)*. IEEE. 2020, pp. 1–5.

[11] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[12] Dibakar Raj Pant et al. “Recurrent neural network based bitcoin price prediction by twitter sentiment analysis”. In: *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*. IEEE. 2018, pp. 128–132.

[13] V Prakruthi, D Sindhu, and S Anupama Kumar. “Real time sentiment analysis of Twitter posts”. In: *2018 3rd international conference on computational systems and information technology for sustainable solutions (csitss)*. IEEE. 2018, pp. 29–34.

[14] Muhammet Sinan Ba Sarslan and Fatih Kayaalp. “Sentiment analysis on social media utilizing machine learning methods”. In: *Advances in Distributed Computing and Artificial Intelligence Journal*. 2020. DOI: `https://doi.org/10.14201/ADCAIJ202093515`.

[15] CP Selvi and R Pushpa Lakshmi. “SA-MSVM: Hybrid Heuristic Algorithm-based Feature Selection for Sentiment Analysis in Twitter.” In: *Computer Systems Science & Engineering* 44.3 (2023).

[16] Amrita Shelar and Ching-Yu Huang. “Sentiment analysis of twitter data”. In: *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2018, pp. 1301–1302.

[17] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[18] Rasika Wagh and Payal Punde. “Survey on sentiment analysis using twitter dataset”. In: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE. 2018, pp. 208–211.

Reviewers — 2024 CCSC Northwestern Conference

Nathaniel Kremer-Herman	Seattle University, Seattle, WA
Chadd Williams	Pacific University, Forest Grove, OR
Zachary Dodds	Harvey Mudd College, Claremont, CA
Paul J. Bonamy	Washington State University, Vancouver, WA
Anna Ritz	Reed College, Portland, OR
Alain Kägi	Lewis & Clark College, Portland, OR
Anonymous Reviewers	