The Journal of Computing Sciences in Colleges

Papers of the 38th Annual CCSC Southeastern Conference

November 1st and November 2nd, 2024
Furman University
Greenville, SC

Abbas Attarwala, Editor
California State University, Chico

Adam Lewis, Regional Editor
Athens State University

Volume 40, Number 5

November 2024

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 Southeastern Conference

Welcome to the 38th Southeastern Regional Conference of the Consortium for Computing Sciences in Colleges. The CCSC:SE Regional Board welcomes you to Greenville, SC, the home of Furman University, for our sixth visit to this beautiful campus. The conference is designed to promote a productive exchange of information among college personnel concerned with producing quality computer-oriented curricula as well as using effective educational methods to teach computer science. It is intended for faculty as well as administrators of academic computing facilities, and also for students to participate in activities that promote computer science. We hope that you will find something to challenge and engage you at the conference!

The robust conference program is highlighted by four sessions of three tracks each, including engaging guest speakers, workshops, a panel discussion, student posters, a nifty assignment session and three sessions of high-quality refereed papers. We received 46 papers this year of which 14 were accepted to be presented at the conference and included in the proceedings – an acceptance rate of 30

Two exciting activities are designed specifically for students – a research contest and an undergraduate programming competition, with prizes for the top finishers in each.

We especially would like to thank the faculty, staff, and students of Furman University for their help in organizing and publicizing this conference. Many thanks also to the CCSC Board, the CCSC:SE Regional Board, and to a wonderful Conference Committee, led by Conference Chair Dr. Kevin Treu, programming contest coordinator Dr. Andy Digh, and research contest coordinator Dr. Fahad Sultan. Thank you all so much for your time and energy.

We extend our deepest appreciation to our partners, sponsors, and vendors. Please take the time to go up to them and thank them for their contributions and support for computing sciences education – *CCSC National Partners*: Rephactor, ACM2y, CodeGrade, GitHub, and Codezinger. Our thanks as well to our *Sponsoring Organizations*: CCSC, ACM-SIGCSE, and Upsilon Pi Epsilon.

We could not have done this without several excellent submissions from authors, the insightful comments from a great team of 34 reviewers, and the support from our editor, Dr. Adam Lewis. Thanks to all of you for helping to create such a strong program for this year's conference.

<div align="right">

Karen Works
Florida State University
Chair, CCSE:SE Regional Board and Regional Representative

</div>

2024 CCSC Southeastern Conference Steering Committee

Stephen Carl, Publicity Chair .... Sewanee: The University of the South
Jonathan Cazalas, At-Large Member .... Florida Southern College
Andy Digh, Programming Contest Director, 2025 Site Chair .... Mercer University
Jean French, Local Registrar, 2023 Site Chair .. Coastal Carolina University
Kala Kennemore, Local Arrangements Chair .... Furman University
Adam Lewis, Program Co-Chair .... Athens University
Karen Works, Chair .... FSU Panama City
Tania Roy, Treasurer .... New College of Florida
Fahad Sultan, Research Contest Director .... Furman University
Kevin Treu, CCSC Southeastern Rep., Program Co-Chair, 2024 Site Chair Furman University

A Packing List for a Trip Into Cyberspace*

Keynote

Carmen Pancerella
Principal Member of Technical Staff Sandia National Laboratories

We live in a world that is connected 24/7. We live in a world where the average person spends over 5 hours per day on a phone and the average teenager spends over 8.5 hours on the phone. We live in a world where data lives in the clouds. We live in a world where an average person may have 8 to 13 devices.

We live in a world where an automated scripts or programs can affect millions of devices at the same time. We live in a world where these automated scripts and programs are also used by malicious actors. We live in a world where malicious actors are targeting government systems, critical infrastructure, and industry networks. We live in a world where the President of the United States is worried about Artificial Intelligence.

How do we prepare students for this world?

Addressing Barriers for Transfer Into University Computer Science Programs for Community College Students*

Panel Discussion

Cara Tang[1], Adam Wade Lewis[2], Karen Works[3]
[1]Computer Information Systems
Portland Community College, Portland, OR 97219
cara.tang@pcc.edu
[2]Department of Mathematical, Computer, and Natural Sciences,
College of Arts and Sciences,
Athens State University, Athens, AL 35611
Adam.Lewis@athens.edu
Computer Science Department
College of Arts and Sciences
Florida State University-Panama City, FL 32405
keworks@fsu.edu

1   Summary

Transferring from a community college to a university computer science program presents many barriers to the student. Many students face personal barriers balancing education with family and work responsibilities. Some of these students wait before continuing on and face challenges in the transition back into the academic environment after many years.

Academically, students transferring from the community college face barriers related to credit transfer, course equivalency, and prerequisite gaps. Without clear articulation agreements, students face the problem of not all credits transferring, especially specialized or technical courses that have no matching

---

university courses. Students often face concerns regarding course equivalency with perceived differences in course content and rigor.

Adult non-traditional students are an important under represented market segment in higher education as small institutions struggle to address questions of financial exigency and demographic changes in this population. Institutions need to understand the demographics and needs of this group of students are very different than the 18-22 year-old age group of traditional college students.

2 Cara Tang

Portland Community College (PCC) is the largest institution of higher education in the state of Oregon and offers a variety of computing education programs, including both those dedicated to preparing students for transfer into a Bachelor's program (Computer Science), and those designed to help students get a job (Computer Information Systems, Cybersecurity, Network Administration, and Web Development and Design). However, even many students in the career-oriented programs intend to transfer to a four-year institution. The ideal transfer pathway is a 2 + 2, where a student spends two years at the community college, likely receiving an Associate's degree, and then two years at a four-year college or university, receiving a Bachelor's degree after a total of four years. Unfortunately, a 2 + 2 is more the exception than the rule.

PCC's Computer Science program is designed to align with Portland State University's Computer Science program, and theoretically it offers a smooth transfer path. However, since many PCC students require additional math, reading, or writing classes, the pathway may be a 3 + 2 instead of a 2 + 2. A further stress point is that PCC's program does not align as well with CS programs at other Oregon institutions students may be interested in, such as Oregon State University and the University of Oregon.

On the other hand, PCC's career-oriented computing programs are highly technical, with the goal of giving students enough skills in two years to get a job in the IT industry, while at the same time, aligning with as many local and regional four-programs as possible in order to keep the door open for students who may choose to pursue a Bachelor's degree. These career-oriented programs have a higher technical component and a lower general education component than transfer programs, and often result in a 2 + 3 model for students who choose to transfer.

Despite many barriers, a positive trend in support of community college transfer students in career-oriented computing programs has been the growing number of Bachelor of Applied Science (BAS) programs in computing disciplines. These programs tend to accept more of the credits that students obtain at the community college, and in some cases, even accept an Associate of Ap-

plied Science degree as a block of two years worth of credit, giving a transfer student immediate junior standing.

3   Adam Wade Lewis

Athens State University is a two hundred year old institution that has been an upper-division only university for over fifty years. So the needs of the adult learner and transfer student are paramount to our institution. The care and support of these students is very different than the usual population of 18-22 year-old students that form the traditional body of college students recruited by our institutions.

For the computing disciplines, our efforts at Athens State University have been deeply focused on the creation and tuning of articulation agreements. This process is complicated by the focus in our community college partners on terminal associate degrees and their pivot towards certificates and micro-credentials. Students transferring with those credentials find themselves with many credit hours that do not transfer and often discover the unfortunate need for two to four additional semesters to address what we call the "general education impedance mismatch".

This is why the work on curriculum guidelines happening within organizations such as ACM2Y is so important. These organizations are putting great effort into harmonizing the curriculum guidelines between the two year and four year institutions. Understanding and aiding this effort is critical to any pivot towards better serving adult learners.

4   Karen Works

The FSU online Computer Science program is an ABET accredited program which accepts students who have completed a minimum of 52 hours of credit at FSU, or an A.A degree. We serve a diverse nontraditional student population and support students around the globe.

Beyond the traditional issues of articulation agreements between schools, we have the additional requirement of ensuring that our transfer students are prepared for the rigor of an ABET accredited program. We work closely with many community colleges and universities throughout the country to ensure that these students make successful transitions.

FSU is dedicated to our non-traditional online students and has developed many resources to ensure that anything a face to face student can do is available for our online students as well.

5   Biographies

Cara Tang is a faculty member in the Computer Information Systems Department at Portland Community College in Portland, Oregon, and leads PCC's Cybersecurity program. She is the Chair of ACM2Y, an ACM group focusing on computing education in two-year programs; Past Chair of the ACM's Committee for Computing Education in Community Colleges; and a Member of the ACM Education Board.

Adam Wade Lewis is an Associate Professor of Computer Science and Program Coordinator for Computer Science and Information Technology at Athens State University, in Athens, Alabama. They are actively involved with transfer advising and academic advising process working with both community college transfer students and transfer students from other senior institutions.

Karen Works is an Assistant Teaching Professor of Computer Science in the Computer Science Department at the Panama City campus of Florida State University.

Incorporating Emerging Cybersecurity Topics in Computing Courses[*]

Conference Tutorial

Cara Tang[1], Sidd Kaza[2], Blair Taylor[2]

[1] Portland Commmunity College

`cara.tang@pcc.edu`

[2] Towson University

`skaza,btaylor`@towson.edu

The mission of the National Cybersecurity Curriculum Task Force (`https://cyberedtaskforce.org/`) is to catalog and create high-quality and relevant curricula on emerging cybersecurity topics, mapping to curricular and workforce guidelines, and make them freely available. This mission is accomplished with the following goals: (1) Conduct a comprehensive search of available curricula in cybersecurity repositories, directories, and among the community. (2) Perform gap analysis to identify high-need areas to create a cyber-ready workforce. (3) Develop high-impact, high-value curricula for the community.

Completing its final year of NSA funding in 2024 through the NCAE-C program (National Centers of Academic Excellence in Cybersecurity), this project has iterated through the three goals in multiple cycles. This tutorial will share the results of the project, demo some of the curriculum produced by the project, and offer opportunities for audience members to explore the cybersecurity curriculum, housed on CLARK (https://www.clark.center/), and ways it can be used to incorporate cutting-edge cybersecurity content into their own courses.

CLARK is the largest platform that provides free cybersecurity curriculum. It is home to high-value, high-impact cyber curriculum created by top educators and reviewed for relevance and quality. This tutorial will explore the curriculum available on CLARK, which includes materials appropriate for cybersecurity courses, as well as materials appropriate for incorporating cybersecurity into any computing courses (IT, CS, IS, etc.). Special focus will be given to curriculum that has been developed through the National Cybersecu-

---

rity Curriculum Task Force project, which covers emerging cybersecurity topics such as zero trust security, ransomware, quantum resistant cryptography, and software supply chain security.

Using a Distinctive Curricular Design Process for Liberal Arts Computing Programs[*]

Conference Tutorial

Jakob E. Barnard[1], Grant Braught[2], Janet Davis[3],
Amanda Holland-Minkley[4], David Reed[5], Karl Schmitt[6],
Andrea Tartaro[7], James Teresco[8]

[1]University of Jamestown, Jamestown, ND 58405
`Jakob.Barnard@uj.edu`

[2]Dickinson College, Carlisle, PA 17013
`braught@dickinson.edu`

[3]Whitman College, Walla Walla, WA 99362
`davisj@whitman.edu`

[4]Washington & Jefferson College, Washington, PA 15317
`ahollandminkley@washjeff.edu`

[5]Creighton University, Omaha, NE 68178
`DaveReed@creighton.edu`

[6]Trinity Christian College, Palos Heights, IL 60463
`Karl.Schmitt@trnty.edu`

[7]Furman University, Greenville, SC 29690
`andrea.tartaro@furman.edu`

[8]Siena College, Loudonville, NY 12211
`jteresco@siena.edu`

As part of its forthcoming article in the Curricular Practices Volume with the new ACM/IEEE-CS/AAAI Computer Science Curricula guidelines (CS2023)[1], the SIGCSE Committee on Computing Education in Liberal Arts Colleges (SIGCSE-LAC Committee) has developed guidance, informed by its sessions at recent CCSC and SIGCSE conferences, to help with the design and/or revision

---

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

[1]`https://csed.acm.org`

of CS curricula in liberal arts contexts [1]. The committee's earlier work found that liberal arts and small colleges approach the design of their computing curricula in unique ways driven by institutional mission or departmental identity. This impacts how faculty at these colleges adopt curricular guidelines. Curricular guidelines like CS2023 inform curriculum design but are balanced with the vision for a program, departmental strengths, locale, student populations, and unique academic experiences. The desire to craft distinctive curricula, combined with the size of curricular recommendations, requires an assessment of trade-offs between achieving full coverage of curricular recommendations and a school's other priorities. SIGCSE-LAC's guidance encourages faculty to reflect on their programs and the role of CS2023, beginning with their institutional and departmental priorities, opportunities, and constraints.

This session will introduce participants to SIGCSE-LAC's guidance to consider curricular development in the context of the unique features of their programs and institutions. Following an overview and brief discussion of CS2023, participants will be guided through an abbreviated design process using the latest version of the committee's reflective assessment process. This process is framed by a series of scaffolding questions that begin from institutional and departmental missions, identities, contexts, priorities, initiatives, opportunities, and constraints. From there, participants will be led to identify design principles for guiding their curricular choices, including the CS2023 recommendations. Examples gathered from the committee's previous CCSC and SIGCSE sessions will be available to help articulate identity and program design principles, which will then be used to identify distinctive program-level learning outcomes. A spreadsheet tool that is being developed to aid in the shaping of curricular choices will be demonstrated. Participants will leave the session with a better understanding of how CS2023 can impact their programs and instruction on how to use the SIGCSE-LACS Workbook outlining our curriculum design process with their departments. Participant feedback will be gathered and used to refine the committee's guidance.

Acknowledgements

Presenter Biographies

One or two of this session's eight co-authors will serve as presenter(s)/facilitator(s).

Jakob E. Barnard is an Associate Professor, Chair of the Computing, Design, & Communications Department, and Director of Online Technology

Programs at the University of Jamestown. He is a facilitating member of the SIGCSE-LAC Committee, and his research involves how curricula have been integrated into Liberal Arts Computing programs. **Grant Braught** is a Professor of Computer Science at Dickinson College. He is a facilitating member of the SIGCSE-LAC Committee. He has organized committee events focused on curricula and published widely on CS education issues, particularly within the liberal arts. **Janet Davis** is Microsoft Chair and Professor of Computer Science at Whitman College, where she serves as the department's founding chair. She co-organized SIGCSE pre-symposium events in 2020 and 2021 on behalf of the SIGCSE-LAC Committee. **Amanda Holland-Minkley** is a Professor of Computing and Information Studies at Washington & Jefferson College. Her research explores novel applications of problem-based pedagogies to CS education at the course and curricular level. She is a facilitating member of the SIGCSE-LAC Committee. **David Reed** is a Professor of Computer Science and Chair of the Department of Computer Science, Design & Journalism at Creighton University. He has published widely in CS education, including the text *A Balanced Introduction to Computer Science*, and served on the CS2013 Computer Science Curricula Task Force. **Karl Schmitt** is Chair and Associate Professor of Computing and Data Analytics at Trinity Christian College. He has served on the ACM Data Science Task Force and various Computing, Technology, and Mathematics Education committees for the MAA, ASA, and SIAM. His interests explore data science education, and interdisciplinary education between computing, mathematics, data, and other fields. **Andrea Tartaro** is a Professor of Computer Science at Furman University. Her computer science education research focuses on the intersections and reciprocal contributions of computer science and the liberal arts, focusing on broadening participation. She is a member of the SIGCSE-LAC Committee, and has published and presented in venues including the CCSC and the SIGCSE Technical Symposium. **Jim Teresco** is Chair and Professor of Computer Science at Siena College. He has been involved in CCSC Northeastern for over 20 years and currently serves as board chair, and has been involved with the SIGCSE-LAC Committee for 5 years. His research involves map-based algorithm visualization.

References

[1]   Amanda Holland-Minkley et al. "Computer Science Curriculum Guidelines: A New Liberal Arts Perspective". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: ACM, 2023, pp. 617–623. ISBN: 9781450394314. DOI:

10.1145/3545945.3569793. URL: https://doi.org/10.1145/3545945.3569793.

Tips and Tricks for Developing Successful Technology Projects for Introduction to Programming and Outreach*

Conference Workshop

Anca Doloc-Mihu, Cindy Robertson, Cengiz Gunay
Department of Information Technology
Georgia Gwinnett College, Lawrenceville, GA, 30043
{adolocmihu,crobertson2,cgunay}@ggc.edu

There is a great need for a diverse and capable workforce to maintain our nation's excellence in science, technology, engineering, and mathematics (STEM). In particular, women and minorities are severely underrepresented in the information technology (IT) fields. In addition to a standard curriculum, outreach activities have been successfully used by many institutions to attract more students to IT. Our Technology Ambassadors Program (TAP) at Georgia Gwinnett College (GGC) has been a successful IT outreach program for eleven years, training a diverse population of college-level students to develop and conduct engaging and fun technology outreach projects to audiences ranging from K-12 students to general education college classes at GGC.

More recently, we have found that these projects [1], originally designed as outreach activities, are also very good resources for introducing basic programming concepts in our classrooms. Our previous work has shown that when our students use block coding as a means to slowly and easily introduce coding concepts, they are more prepared for text-based programming skills. Using block coding to program a game or activity to learn algorithmic thinking is not only more fun, but more importantly it is a better way to develop the step-by-step process of writing coding instructions and is more enticing to students. This flow allows them to understand and be able to apply basic computing concepts and enjoy the learning process.

In this workshop, we will introduce the audience to different types of en-

---

*Copyright is held by the author/owner.

gaging projects that faculty and students perform at our institution and to our TAP repository website where we host freely available projects. We aim to spark lively discussions about how to best introduce programming skills to engage audiences of any age and establish new collaborations with audience members.

References

[1] Cindy Robertson and Anca Doloc-Mihu. Understanding college level student learning of basic programming at an open access institution. In *Proceedings of the 2023 ACM Southeast Conference (ACMSE)*, page 26–32, NY, USA, 2023. ACM.

Message Passing and Shared Memory Projects[*]

Nifty Assignment

William Kreahling
Department of Mathematics and Computer Science
Western Carolina University, Cullowhee, NC 28723
`wkreahling@wcu.edu`

Abstract

Concurrent programming is an important topic that involves many concepts that learners struggle to understand. In my class we go over topics, involving processes, process creation, threads, message passing, shared memory, race conditions, deadlock and many others. Learners need practice with these concepts to fully understand the issues and the synchronization primitives that can be used to write safe concurrent programs.

Finding projects that learners can successfully complete within a small timeframe can be difficult. Several years ago I wrote two assignments focusing on the aforementioned topics that could be solved using a variety of synchronization primitives. In the past learners have completed these assignments using Java or Rust.

I created a message passing project where students simulate 'Buffon's needle'. In this project learners learn the basics of threads and threadpools and use message passing techniques for process communication. I also created another assignment, based on the concurrency problem first described by Suhas Patil in 1971. In this assignment learner's use shared data and synchronization primitives like semaphores and condition variables.

Material

No materials are required beyond a programming/runtime environment. Many students choose to use an integrated development environment (IDE), but it is

---

not required. This paper goes over details of both assignments, but more explicit details and rubrics will be provided upon request. Some of the challenges learners will face will vary depending on the programming language used. For example, shared memory in Java is as easy as passing references to objects and learners can accidentally use shared memory without realizing it, while in Rust, the Rust compiler will generate warnings and errors if shared memory is not set up 'safely'.

Message Passing Assignment

The first assignment is based on Buffon's Needle. Buffon's Needle is an experiment where one drops needles onto a floor that have equally spaced lines on it, if a needle crosses a line it is a 'hit' and if the needle does not cross the line it is a 'miss'. Using data from many independent needle drops, one can estimate the value of PI. The learner is given an assignment handout, with a supplied mathematical formula, to help them estimate PI from the experiment data.

One way to estimate PI is using the Formula $\frac{2N(hits\ +\ misses)}{L(hits)}$, where N is the number of total number of needles to drop in the experiment and L is the distance between the lines. A 'hit' occurs when a dropped needle crosses a line. To simplify this experiment, the program should ensure the size of the needle is less than the distance between the lines.

Learners recreate this experiment utilizing multiple threads via threadpools. Multiple 'worker' threads are responsible for simulating multiple needle drops while the main thread collects all the data and estimates PI. Message passing is used for communication between threads. For the experiment to accurately estimate PI, learners need to perform many hundreds of thousands of needle drops and collect data, which lends itself very well to concurrent programming.

As part of this assignment the learner's code must demonstrate that their experiments produce correct results and show that their solution is done correctly using message passing and not shared memory. Learners must also show that the data from each 'worker' thread is gathered by the main thread as each thread finishes and not necessarily in the order the threads were started. Learners concurrent solutions must execute faster than a single threaded solution. As part of this assignment learners are not allowed to explicitly use any built in constants for PI.

One weakness of this project is that is requires a shared testbed to fairly compare solutions, so that everyone has the same resources such as CPU quantity and capability, memory, process quotas, etc. This shared testbed could be a departmental server or a virtual machine.

Shared Memory Assignment

For my shared memory project I created an assignment based on the concurrency problem first described by Suhas Patil in 1971. Learner's are given a scenario involving limited resources where those resources are needed by multiple consumers. In this scenario, there is a single distributor of these resources. In the project handout, a common, flawed solution that leads to deadlock is covered. Learners are then given a set of guidelines and restrictions for solving the problem. Correct solutions must used shared memory and prevent deadlock and starvation. To solve this problem students need to correctly use synchronization primitives. In the past, learners have used binary and counting semaphores, binary semaphores and condition variables, or monitors and condition variables.

In this assignment there are three resources, three consumers, one distributor, and N number of messengers. Each consumer has an unlimited supply of one of the three resources, but needs the other two to perform work. No two consumers can have an unlimited supply of the same resource. The consumers, messengers, and distributor all run in their own threads. All the threads must work together, run concurrently, and can only communicate using shared memory. When processes have no useful work to do, they should not be consuming CPU time.

Each consumer needs all three resources to perform any work. In this scenario, each consumer will try to acquire the two resources it lacks then prepare them and perform work. 'Preparing resources' and 'work' are simulated by having consumers sleep for a random amount of time. Each consumer may only get the other resources it needs from shared memory.

The distributor has an endless supply of all three resources and its job is to distribute supplies to a shared memory location. The distributor will randomly choose two of the three resources and place them in a shared memory location, the distributor will make sure that the two resources are never of the same type. The distributor cannot distribute any more supplies until the two resources previously delivered have been removed from the shared memory. Each resource placed in shared memory will be guarded by a separate and distinct lock, such as a semaphore, or mutex.

Once consumer processes have obtained a resource from shared memory, they will never give it up, and resources are consumed during 'work' and consumer threads will not communicate with other consumer threads. Once a consumer has all three resources to perform work, it will signal the distributor that it can deliver more supplies to the shared memory location.

Messenger threads are allowed to communicate with consumer threads, access shared memory, and move the resources once the distributor has placed them in shared memory. How many messengers are needed and their exact

role is up to the learner to determine.

For this assignment, students must ensure this program allows all consumer processes to perform work 'at the same time' without deadlock, livelock, starvation, or explicit polling, until the program is terminated by the user.

As with the message passing assignment it is stipulated that learners must write their code so that it demonstrates the code works correctly and that threads run concurrently. This assignment can lead to a greater understanding of concurrency and the topics surrounding it as well as a deeper understanding of the tools available to implement concurrent programming, including threading, shared data, shared memory, and the synchronization primitives that are available for use in most programming languages.

This assignment is not without weaknesses. There are artificial limits imposed on the assignment, so the logic of how the distributor thread and consumer threads are allowed to communicate and share resources must be made **very** clear in the instructions.

Anecdotally, I have had multiple learners over the years tell me how much they have enjoyed this assignment.

Three Phase - Adversarial Search - Tile Games[*]

A Nifty Assignment

Karen E. Works
Computer Science
Florida State University
Panama City, FL 32405
`keworks@fsu.edu`

Abstract

With the advent of chatGPT and Copilot I find that students are not delving deep enough into the implementation of search approaches. To combat this, I decided to implement a three-phase adversarial search project. After lectures on adversarial search approaches and implementation examples, students are given code to a user versus user basic tile game. They are informed of the three phases of the assignment with the goal of encouraging students to understand that they are expected to be able to read and understand an adversarial search logic. In the first phase, all students use the user versus user basic tile game to implement a computer versus user basic tile game app that utilizes an adversarial search. In the second phase, students create and implement their own computer versus user basic tile game app by changing the rules on how the tile game is won and what a valid move is. In the third phase, students are given a timed 10 minute quiz where they are given code for a tile game and the rules for how the game is won and valid moves. The students must identify if the adversarial search is properly implemented and if not then what logic is not correct.

Materials

The only software requirements for this assignment is an integrated development environment (IDE). All materials presented and coding examples are

---

available upon request.

Assignment

The basic tile game starts with a randomly generated three by three grid where three tiles are marked with '1', three tiles are marked with '2', and 2 tiles are marked with '9' (Figure 1).

Figure 1: Start Game

During a turn, a player may move any tile into the one blank space on the board. The possible moves are up, down, left, and right and are bounded by where the blank space lies on the board. For example, in the start scenario above, a player may move a tile up (Figure 2a) or left (Figure 2b) into the blank space on the board.

Figure 2: Example Valid Moves

(a) Up

(b) Left

One player is assigned '1' and the other player is assigned '2'. The goal is to get three vertical tiles, three horizontal tiles (Figure 3b), or either diagonal (Figure 3a) to match their assigned number.

The first two phases provide the students an opportunity to learn how to implement an adversarial search and seek help when required. Phase 3 encour-

Figure 3: Example Wins

(a) 1 - Diagonal

(b) 2 - Vertical

ages students to seek a deeper understanding of the material and to proactively ask questions. After having success with this approach, in upcoming semesters I am considering having the students do a synchronous live code review in lieu of an online quiz.

Metadata

Summary	Students are given a user versus user basic tile game. - Phase 1: Students implement a computer versus user basic tile game app that utilizes an adversarial search. - Phase 2: Students create and implement their own computer versus user basic tile game app by changing the rules on how the tile game is won and what a valid move is. - Phase 3: Students are given a timed 10 minute quiz where they are given code for a tile game and the rules for how the game is won and valid moves. The students must identify if the adversarial search is properly implemented and if not then what logic is not correct.
Topics	Apply classical Artificial Intelligence techniques, such as search algorithms, minimax algorithm, and greedy algorithms to solve problems
Audience	Mid CS2
Difficulty	**Medium:** Given basic Python application that supports tile game, implement AI search algorithms.
Strengths	Allows students to learn classical Artificial Intelligence techniques, such as search algorithms, minimax algorithm, and greedy algorithms and understand how to use these techniques to solve a problem.

Metadata(cont)

Weaknesses	Students must read and understand the user versus user basic tile game code. Reading code they have not written may be a challenge for some students.
Dependencies	Starting version of the user versus user basic tile game.
Variants	Starting version of the goals of the tile game can be adapted to develop different versions of the game.

Context-Driven English to Japanese Translation with Visitors[*]

Johnathan Dewey, Chris Alvin
Computer Science Department
Furman University
Greenville, SC 29613
{ johnathan.dewey, chris.alvin[†]}@furman.edu

Abstract

We present a mechanical translator that uses context to drive the transfer of phrase representation from English to Japanese, thus effectively producing a translation. Our translator consists of three distinct steps. The first step takes raw English text, tokenizes it, and classifies each token according to its part of speech. These tokens act as terminals in a *sentence tree* that is constructed using a bottom-up technique. The next step works to manipulate the internal nodes (non-leaf) nodes of the sentence tree to reflect an ordering consistent with the sentence structure of the Japanese language: subject-verb-object in English to subject-object-verb in Japanese. The last step involves translating terminal nodes in the sentence tree from English to Japanese, verifying conjugation, and ensuring correct Japanese particles. We verified the effectiveness of our proof-of-concept translator by testing for usage of proper nouns, conjunctions, and all variations of tense features.

1 Introduction and Background

Computational translation can be inaccurate. Therefore, we propose a technique that uses context of language to aid in translation. Words of a language

are normally broken into distinct parts of speech (POS) which are classes of lexical items that have a specific function within the language. Sentence formulation is broken into three distinct parts which determine the sequence in which certain words are uttered: subject, objects, and verbs. These two observations of speech are good indicators of context that can be used by a translator to make decisions during the translation process.

There exists a noticeable schism in schools of thought related to computational translation. On one hand, there are advocates for Statistical Machine Translation (SMT) which employs a linguistic corpus: a large database that contains linguistic lemmas and data such as its frequency in specific texts. Therefore, SMT uses empirical data on the frequency and context of words to provide a more cohesive and context-sensitive translation [3]. On the other hand, there are advocates for Neural Machine Translation (NMT) which uses Machine Learning (ML) techniques to train a model to learn and predict the equivalent meaning of phrases between languages. With fervorous advancements among ML, many think that it is objectively better.

SMT is useful in the scope of solving context-dependent translation problems and thus an effective method of translation. The main feature of SMT is a linguistic corpus that organizes words of a language and relevant linguistic information. One instance of relevant linguistic information is the POS tagging that enables the organization of lexical items into their respective parts—such as nouns, verbs, adjectives, etc. POS tagging provides richer contextual information for a sentence and improves decision making in translation [4]. Additionally, POS tagging enables systems to interpret the structure of sentences without knowing the exact translation [4]. Regularities in phrase structure and POS can be used as context and can make decisions to attain translation without knowing the individual words. This fact is emphasized in the translator where the meaning of words is not needed to drive translation. In fact, translation of words after a Japanese phrase structure is derived is the only instance where meaning of a word is needed.

However, there are some detriments to POS tags, as certain POS can often be redundant and can hinder decision-making. POS like nouns are known to be strong in aiding decision making, while redundant words such as determiners impede the decision making of translation [4]. However, it is theorized in our proposed translator that by treating certain POS as more important through a precedence hierarchy, thus tagging can be mitigated. In such a hierarchy, POS like verbs and conjunctions are treated as superior in contextual meaning than that of a noun or determiner. Therefore, representations of phrases can easily be discernible from the existing POS in the phrase.

When comparing SMT and NMT it is hard to determine if one is objectively better than the other because they both yield accurate results. However, both
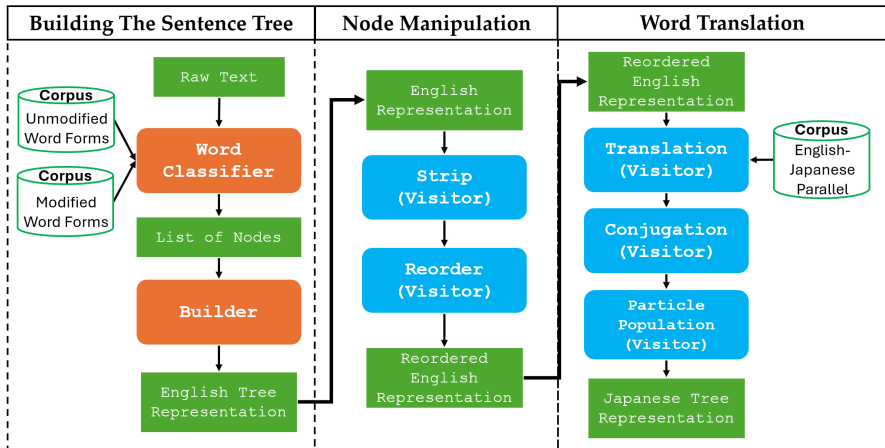
Figure 1: The workflow for context-based English to Japanese translation using sentence trees.

techniques are known to have their own detriments such as NMT techniques can have ill-defined test data of ML models while SMT has issues with management and expansion of a corpus. As an apples-to-apples comparison, Phan and Jannesari [7] employed the use of two SMT's and two NMT's to translate source code from one programming language to another. In their experiment, translation yielded a percent accuracy of 60-90% for SMT and 59-83% accuracy for NMT [7]. When translating source code, it can be asserted that SMT outperforms NMT in their current state [7]. Thus SMT is a relevant approach to mechanical translation, and if SMT can outperform NMT for a simpler abstraction of language, then it is theorized that a natural language translator will yield accurate translations if it employs a SMT system.

This paper describes an SMT-based approach to translation of English to Japanese. As shown in Figure 1, our process works in three phases: construction of a syntax tree we call a *sentence tree*, internal node manipulation on the tree, and finally translation of terminal nodes. All source code is available online[1].

2   Theory: Syntax Trees and X-Bar

Syntax trees in linguistics are the hierarchical organization of lexical items within a phrase, and a traversal of the sentence tree produces the sequential utterance of lexical items of a phrase. Our translator is based on X-Bar Theory,

---

[1]https://github.com/john-dewey/Context-Based-Translator
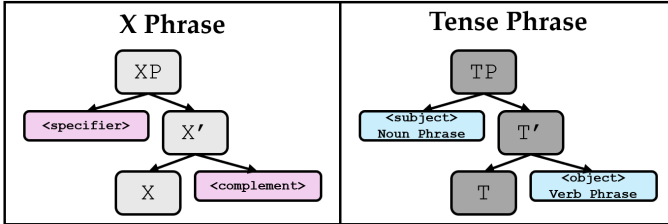
Figure 2: Structure of the X-Bar phrase node [1]. X is the 'head' or POS (Noun, Verb, etc.) that carries the most meaning. The X-bar (X') establishes a close connection between X and its complement; specifier are phrases occurring before the head. Tense Phrase (TP) structure captures tense features and modal auxiliaries ("can," "should," etc.); these features are encoded as the Tense terminal T. T' establishes a close connection between tense and the verb phrase, whereas TP exhibits the placement of a subject before a Verb phrase.

a theory of syntax that creates representations of nuanced phrases; see Figure 2. X-bar aims to make stronger predictions as to what constitutes a grammatical sentence [1]. Their key feature, the X-bar node, is an extra node of abstraction that permits nuanced or "non-flat" phrase structures so that phrases can be independent of each other yet have associations [1].

Another feature of X-bar Theory is the assumption that phrase structure is identical for all POS and all languages. While a bold assumption, it is based on some truth. Many languages share a commonality in phrase structure where the head (lexical item carrying meaning) of a sentence tree shares a more intimate relationship with their complement than specifiers [1]. This fact is reflected in the structure of an X-phrase (XP) node where the first level of the phrase includes a specifier ("I" in "I went home"), and the second level includes a complement ("home" in "I went home"). The complement is placed next to the head ("went" in "I went home") of the phrase to preserve this close relationship, and therefore has the specifier come before the head. X-bar can be expressed as a binary tree [5], thus resulting in efficient translation algorithms.

3   The Translation Algorithm

In this section, we consider the English sentence "She found a job last Friday" which translates into Japanese as watashiwa saigono kinyoubini shigotowo mit-suketa (私は最後の金曜日に仕事を見つけた). This example will serve to communicate our translation algorithm workflow shown in Figure 1; Figure 3 depicts a more detailed sentence-tree perspective as well as the impacts of employing several visitors over the sentence tree.
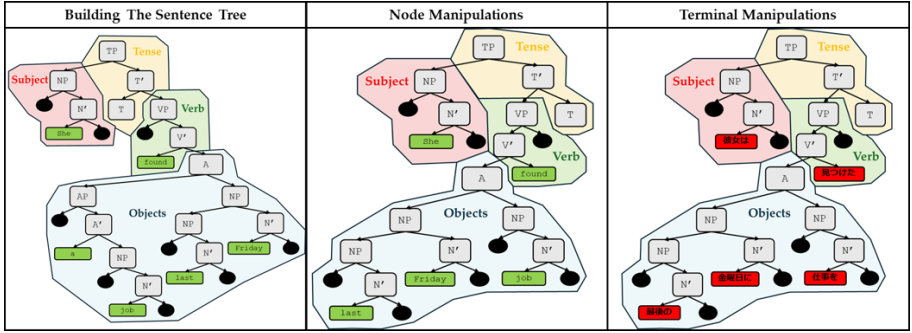
Figure 3: Overview of context-dependent translation: building a sentence tree, manipulating internal nodes, and manipulating terminal nodes.

3.1 Preprocessing

Our first step identifies input words not contained in the *primary corpus* [2]: a simple database of English words and their POS. One common discrepancy is the use of contractions to exhibit tense features; e.g., "won't" is a contraction of the tense feature "will" and a negative "not." Contractions do not exist in the same capacity in the Japanese language; thus, they are split into two distinct lexical items.

Corpus lookup. Each word is looked up in our primary corpus [2] containing only the lexical information for concrete word forms. For modified word forms like "went" (modified from "go") or "working" (modified from "work") they will fail the initial lookup and perform another search into a secondary corpus [2]: a map from modified word forms to their unmodified word form. If an English word exists in the corpus, then it can be represented fully in the resulting tree; however, if lexical items like proper nouns are not in the corpus, the translator assumes are nouns.

We observed that word encodings in the secondary corpus contained discrepancies; e.g., continuous forms of verbs such as "working" would be encoded as an adjective. To address this issue, the translator implements a context verifier that ensures that POS makes sense contextually. For example, adjectives must reference a noun or another adjective. In cases where such requirements are not met, nodes are re-encoded to find the correct POS that should exist.

3.2 Building the Sentence Tree

The previous steps provide a sequence of terminals in our sentence tree. That is, we create a sentence tree using context derived from the type of nodes present. As shown in Figure 3, a resulting sentence tree stores the subject,

verb, and objects as children of the tense features in a sentence. The structure uses vertex nodes (NP, N') to create hierarchical relationships between phrases, and uses terminals to represent the lexical items of a sentence.

Building the tense phrase. Our goal in this step is to make a sentence tree out of a list of nodes. These nodes are sent to the root node of a sentence tree which is the TP node (see Figure 2). In X-Bar Theory, TP is a phrase node that incorporates the requirements for a complete sentence.

A TP node has a left child that represents the subject of the sentence. The right child is known as the T-bar (T') node and allows for its right child to contain a verb phrase that represents the verb and object of a sentence. The left child is known as the tense (T) node because it stores all the tense features such as present/past or modal auxiliaries, of the sentence. In our example (Figure 3), the subject is a noun phrase containing the pronoun "She"; "She" is a left-child of TP. The rest of the nodes (found, a, job, last, Friday) go to the right child which is the T'.

In T', the tense information of the sentence is recorded in its left child which is the T terminal. After the recording of tense information, the same nodes (found, a, job, last, Friday) are sent to verb phrase (VP) so that the verb ("found") can be separated from the objects (a, job, last, Friday) of a sentence. We know it is a VP because the X-Bar structure of nodes states that the right child of T' is the VP required for a complete sentence [1]. VP then discerns the location of the verb in relation to the remainder of the sentence. The verb "found" does not have any specifiers, or words preceding it, so we can pass on to the verb bar (V') level. V' separates the verb terminal and the remaining sequence of nodes and encodes "found" as the V's left child. Once the verb is encoded at the V' level, the remaining objects are sent to an adjunct node.

Building the adjuncts. Adjunct nodes act as a separator between sub-phrases (left panel Figure 3). The direct object of the verb "found" is "a job," so this constituent phrase is the left child of the adjunct so that it is next to the verb in the context of a sentence tree's traversal. This position is known as the direct object because it is the complement of the verb in the sentence. The phrase "last Friday" is an adjunct phrase, so it goes to the right child of the adjunct node so that it comes last in the context of a sentence tree's traversal.

Constituent phrases that reference the adjunct node are determined through an adjunct identifier, which breaks down individual constituent phrases. "a job" consists of an article and a noun which is recognizable constituent phrase to the identifier; similar for "last Friday." These phrases are encoded as separate, so as not to put them directly under the same hierarchy and inadvertently make the phrases dependencies of each other.

3.3 Manipulating the Sentence Tree through Visitors

The remainder of the translation algorithm manipulates a sentence tree to conform to the structure of a Japanese sentence. Each manipulation is implemented as a unique *visitor*: an object-oriented design pattern that facilitates flexible, easy, and abstract visitation over elements in a data structure [6]. Visitors are used to traverse the sentence tree and make decisions regarding the structure of the sentence tree and the information held in its nodes.

3.3.1 Internal Nodes

The goal of these visitors is to preprocess a sentence tree from a "Subject-Verb-Object" (SVO) English sentence structure to a Japanese "Subject-Object-Verb" (SOV) sentence. To do so we (1) remove unnecessary lexical items and (2) reorder the sentence tree to achieve a Japanese word order—relying on POS context to make decisions.

Strip. Japanese lacks a clear future tense, while in English the word "will" is very indicative of an action happening in the future. In Japanese, future tense is not discernible from verb conjugations and use the same conjugations as present tense. Therefore, in terms of conforming to the Japanese sentence structure, we remove any words that do not exist in the same context as English words. Similarly, for articles, there is no definite ("the") or indefinite ("a") article for nouns in Japanese. Therefore, these unnecessary words must be removed from noun phrases. As shown in Figure 3, the *strip* visitor will result in a sentence tree that reads "She found job last Friday."

Reordering. Sentence phrase order is arguably the largest discrepancy between English and Japanese translation: SVO versus SOV, respectively. This means that the structure of nodes and their children must match the traversal of a Japanese sentence tree. The reordering visitor acquires a SOV structure. For example, upon visiting a verb phrase, the visitor switches the node's children so that the verb comes last. In some cases, a more complex approach is required: a node's children may be reordered based on conditionals influenced by specific situations, instead of generalizations based on POS. For example, when a visitor visits an adjective phrase node, and it detects a "not" phrase attached to its specifier (left child), it reorders the node's children so that the "not" phrase comes after the adjective as it does in Japanese.

In panel 2 of Figure 3, "She found job last Friday," the T' node is reordered so that its right child is the T terminal with tense information and should come last. Then, both the VP and V' node's children need to be reordered so that the verb comes after the objects. Finally, the objects need to be reordered so that the direct object "job" is next to the verb "found."

3.3.2 Translation Visitors: Terminal Nodes

We continue walking through our workflow by consider the right panel of Figure 1. The remainder of the visitors manipulate the terminals of the sentence tree to achieve the grammatically correct representation of the Japanese syntax. This is done through (1) transferring English text to Japanese, (2) verifying or constructing Japanese conjugations, and (3) verifying or constructing Japanese particles. Since particles have a post-fix position in relation to a word, they are ordered as the last task before achieving a translation.

Translation. At each terminal in the sentence tree, a visitor accesses an English-Japanese parallel corpus that maps English words to Japanese words. We were unable to identify a free, or even a reasonably costed, well-formed parallel corpus. As a result, we manually constructed a parallel corpus to facilitate testing. This creates an imbalance when considering the primary and secondary English corpora used in sentence tree construction.

Conjugation. In the verification or construction of Japanese conjugations, tense features and other modifiers such as negatives are searched and used as context to produce proper conjugations with the right tense and modality. In the example, the verb "found" is conjugated correctly as mitsuketa (見つけた) through using the parallel corpus.

Japanese verbs are ordered with modifiable stem, a conjugation for tense, and a conjugation for its modality (positive or negative). We thus conjugate verbs and adjectives in the sentence tree to include the stem and tense conjugation, but in the case of a negative "not" connected to its phrase, the tense conjugation is left out and handled by the "not" terminal. For example, "did not find" would be treated as two words "not" and "find." After reordering, this phrase is "find not [past]," which the translation and conjugation visitors would translate to mitsuke naka tta (見つけなかった).

Japanese verb conjugation also has three classes of verbs that require different conjugations. The godan (五段) verbs are the most common and has five possible stem conjugations. The ichidan (一段) verbs are common but have irregular conjugations due to an -iru/-eru (いる/える) verb ending. Finally, the irregular class of verbs contain the words "to do" (suru する) and "to come" (kuru くる). Adjectives have two classes: the adjectives ending in an -i (い) sound and adjectives ending in a -na (な) sound.

Particle population. Particle population visits each constituent phrase's head and decides what particle is appropriate to attach for each phrase. In Japanese, each constituent phrase is assigned some particle which denotes its relationship with both the verb and the sentence. However, using X-bar theory we can discern the location of phrase specific particles. For example, the head of the TP's left child is the subject of the sentence [1]. Therefore, we can recognize "she" (彼女) as the subject and add the particle wa (は) to denote

Table 1: Translations focusing on different tense features and their verb forms.

	Excerpt from Test Data		
ID	English	Japanese	Tense
1	I'll be giving a presentation at the conference.	私は会議でプレゼンテーションを供与している。	Future Continuous
2	I will not be attending the party tonight.	私は今夜にパーティーを出席していない。	Future Continuous
3	I will visit my parents next weekend.	私は次の週末に私の両を親訪ねる。	Future Simple
4	He will not pass the exam without studying.	彼は勉強しているせずに試験を合格しない。	Future Simple
5	She was singing loudly during the concert.	彼女はコンサートの間大声で歌っていた。	Past Continuous
6	I wasn't paying attention to the lecture.	私は講義に注意を払っていなかった。	Past Continuous
7	My friend and I didn't buy anything on our trip.	私の友達と私は私たちの旅行で何でも買わなかった。	Past Simple
8	Rachel didn't tell anyone the secret.	レイチェルは秘密を誰かに言わなかった。	Past Simple
9	I am currently working on a new project.	現在、私は新しいプロジェクトで取り組んでいる。	Present Continuous
10	We aren't going out for dinner this evening.	私たちはこれの晩夕食のため外に行っていない。	Present Continuous
11	He's learned to play the piano.	彼はピアノ遊ぶに習っている。	Present Perfect
12	We haven't seen that movie.	私たちはその映画を見なかった。	Present Perfect
13	My school starts at 8:00.	私の学校は8時で始まる。	Present Simple
14	We always eat dinner together.	いつも私たちは一緒に夕食を食べる。	Present Simple

this function. Furthermore, the head of the right child's phrase in a V' is the object of the verb [1]. Similarly, we can recognize "job" (仕事) as the object and add the particle wo (を) to denote this function.

A final visitor outputs the translated Japanese sentence.

4   Results

The translator was able to produce Japanese translations for 60 English sentences in past, present, future tense, as well as their continuous, simple, and perfect forms. An excerpt from those test cases is provided in Table 1; we discuss these numbered tests below. In addition to all tense features, there are grammatical structures that are accurately represented in Japanese. For example, in test 5, "during" follows the Japanese pattern: Noun + の間. Furthermore, conjunctions in the context of two nouns, such as in test 7, is accurately represented in the Japanese form: Noun + to (と) + Noun.

We also observe correct, nuanced ordering of POS in sentences like test 9. In Japanese, adverbial phrases usually come at the beginning of the phrase, unlike English that usually introduces them after the subject. Owing to the X-bar structure, we can effectively separate "currently" from the subject "I" and reorder it so that genzai (現在), meaning currently, comes first in the sentence.

Our translator does have some deficiencies. Our English-Japanese parallel corpus that stores English and corresponding equivalents in Japanese along with parts of speech was created as a bespoke utility. While not ideal, no

other existing tool provided the nuanced information required by the context-based translation. As with any custom tool or database, this might result in overfitting of the parallel corpus to the test data. For example, in test 8 and 13, the words "Rachel" and "8:00" are examples of words that are translated well due to the overfitting of the bespoke parallel corpus. "Rachel" is a proper noun, and if a proper noun like "John" was used, the translation visitor would encode a null value into its terminal rather than jon (ジョン) in Japanese. Similarly, "8:00" is a concept of time, and the Japanese represent time differently. The parallel corpus returns "8時," but for other cases it would need to follow the format for Japanese time: Hour + ji (時), Minute + bun (分).

Furthermore, we observe that the accurate use of verbs is not being used effectively. For example, in test 11, the phrase "play the piano" does not use the correct word for "play" in Japanese. The current word it translates to asobu (遊ぶ) does not make grammatical sense in the context of a "piano," and instead should be hiku (弾く). This error also occurs in test 10 where "going out" should be the verb dekakeru (出かける) and not sotoni iku (外に行く).

Last, we observe that Japanese particles are not populated effectively. As for translation, it is simple to translate prepositional particles due to their words in the parallel corpus matching the prepositional particles in Japanese. However, for other constituent phrases it was harder to discern the correct particle. This is because Japanese particles have nuanced applications that cannot be discerned on POS context alone. Therefore, this translator assumes the particle ni (に) for certain noun constituent phrases, and the particle wo (を) for objects of the verb. While this assumption accurately portrays some particle usages, it is largely an ineffective feature that can be addressed in future work by integrating an NMT.

5 Conclusions

We have proposed an SMT context-driven translator for translation of English to Japanese. We leveraged X-bar theory to drive our means of representation as sentence trees with greater nuance. X-Bar's inclusion of a bar node helped separate and organize unique features of a word and permitted nuanced tree structures without compromising the sequential ordering of words. The theory was able to accurately represent all the test sentences in both English and Japanese, which is evidence of the translator's utility and emphasizes its universality. This work is an example of the power of linguistics theory and the visitor design pattern implemented over a tree structure. This work also represents a foundation for further exploration in translation, but may also provide a unique opportunity for undergraduate students to employ their knowledge of fundamental data structures and relatively straightforward algorithms.

References

[1] Catherine Anderson et al. *Essentials of Linguistics*. 2nd ed. eCampusOntario, 2022.

[2] Mark Davies. "The 385+ million word Corpus of Contemporary American English (1990–2008+): Design, architecture, and linguistic insights". In: *International journal of corpus linguistics* 14.2 (2009), pp. 159–190.

[3] Julia Hirschberg and Christopher D. Manning. "Advances in natural language processing". In: *Science* 349.6245 (2015), pp. 261–266. DOI: 10.1126/science.aaa8685. eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8685. URL: https://www.science.org/doi/abs/10.1126/science.aaa8685.

[4] Robert M. Losee. "Natural language processing in support of decision-making: phrases and part-of-speech tagging". In: *Information Processing & Management* 37.6 (2001), pp. 769–787. ISSN: 0306-4573. DOI: https://doi.org/10.1016/S0306-4573(00)00061-3. URL: https://www.sciencedirect.com/science/article/pii/S0306457300000613.

[5] MD Nimram et al. "Exploring X- Bar Syntax as Applied to English Syntactic Structures". In: *Greener Journal of Language and Literature Research* 8.1 (2023), pp. 12–28.

[6] Jens Palsberg and C. Barry Jay. "The Essence of the Visitor Pattern". In: *COMPSAC '98 - 22nd International Computer Software and Applications Conference, August 19-21, 1998, Vienna, Austria*. IEEE Computer Society, 1998, pp. 9–15. DOI: 10.1109/CMPSAC.1998.716629. URL: https://doi.org/10.1109/CMPSAC.1998.716629.

[7] Hung Phan and Ali Jannesari. "Statistical machine translation outperforms neural machine translation in software engineering: Why and how". In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Representation Learning for Software Engineering and Program Languages*. 2020, pp. 3–12.

Software Orchestration: A Paradigm for Software Development and Security Assessment using ChatGPT Requirements*

Joseph Elarde, Barry Bruster, Mir Hasan
Austin Peay State
University
Clarksville, TN 37044
elardej@apsu.edu, brusterbg@apsu.edu, hasanm@apsu.edu

Abstract

Software orchestration integrates AI tools like ChatGPT into the software development process, evolving beyond traditional methods. This paper introduces Software Orchestration, blending the concepts of a musical symphony, where a conductor guides an orchestra, with automated processes in computing. Here, AI Neural Networks act as the "orchestra" and the Software Engineer as the "conductor," collaboratively crafting, refining, and executing software. This approach combines human expertise with AI capabilities, enhancing software design, development, validation, and documentation. We explore its principles and applications in software development, supported by nine experimental case studies, highlighting its transformative potential in the software industry.

1 Introduction

The quest for producing software that is exceptional in quality, secure, maintainable, and well-documented is pivotal for technological advancement. Rapidly delivering superior software has become a critical competitive edge for businesses, turning agility in software development from aspiration to necessity.

Traditional software engineering paradigms often struggle to balance rapid development with quality assurance. The integration of Artificial Intelligence (AI), particularly Large Language Models (LLMs) like ChatGPT [9] has introduced 'software orchestration', harmonizing rapid development with quality. Our research explores software orchestration, focusing on prompt engineering for optimal outcomes. We address the following questions:

1. What are the most effective prompts or prompt engineering techniques for software orchestration in design and development phases?

2. How does software orchestration, compared to traditional methods, influence productivity?

This paper is structured as follows. Section 2 introduces our proposed software orchestration process, detailing design, development, documentation, debugging, deployment, and maintenance. Section 3 outlines our evaluation methodologies. Section 4 presents our assessment results. Section 5 discusses the implications of our findings. Section 6 provides background and literature review, situating our research within the broader academic discourse. Finally, Section 7 summarizes our conclusions, contributions, and potential avenues for future research.

2 AI Assisted Software Development

Let's conceptualize AI-assisted software development as a hierarchical structure, comprising four distinct levels of support:

1. **Code Autocompletion:** Representing the foundational level of AI assistance, this involves the IDE offering suggestions for code elements, such as variables and methods, rooted in the immediate context.
2. **Function Insertion (e.g., GitHub Copilot):** Progressing a step further, AI at this level can propose or generate intricate code snippets and/or entire functions. GitHub Copilot exemplifies this tier.
3. **AI Natural Language Assists (e.g., Chat GitHub Copilot):** This level allows developers to converse with AI in natural language to address coding assistance, answer queries, or requesting high-level code generation.
4. **Complete Software Lifecycle Support (Software Orchestration Process):** The pinnacle of AI integration, this level sees AI's involvement in every facet of the software development lifecycle. From the nascent design phase to debugging and maintenance, AI's holistic involvement has the potential to redefine the paradigms of software creation and evolution.

This hierarchical framework underscores the escalating role of AI in software development. It chronicles AI's journey from offering rudimentary code suggestions to its comprehensive engagement in the software lifecycle.

2.1 Software Orchestration Introduction

Software development is undergoing significant transformation with the integration of Artificial Intelligence (AI), particularly through "Software Orchestration." This paradigm likens AI Neural Networks to an "orchestra" and Software Engineers to "conductors" who guide AI in designing, refining, and executing projects. This collaborative synergy enhances software craftsmanship beyond conventional automation. Traditionally, the Waterfall model [4] has dominated software development with its linear, sequential phases: Requirements Gathering, Design, Implementation, Testing, Deployment, and Maintenance. While systematic, its rigidity has led to the adoption of dynamic methodologies like RAD, Spiral, and Agile to meet modern development needs. Software Orchestration modifies this process by retaining requirements gathering and harnessing AI for rapid prototyping and iterative development. It encompasses stages such as Initial Prompt, Reflection, Infrastructure Implementation, Application Implementation, Debugging, Post-Development Review, Deployment, and Maintenance, detailed below.

1. **Initial Prompt:** The initial prompt in Software Orchestration is critical for setting the stage for the application's design and functionality. For example, the "Student Class Planner" application aids faculty in managing graduate students' study programs. The prompt includes a description of the following: Background, Basic Application Design, Modular Design, Database Structure, GUI Framework, Error Handling, Authentication, Concurrency, External Libraries, Deployment, User Roles, Data Validation, Backup and Recovery, Performance Requirements, System Integration, UX/UI Design, Logging, Feedback Mechanism, Training, Documentation, and Scalability. Refer to the GitHub site for a complete example [3].

   Detailed initial specifications are essential for shaping the development process, explaining the roadmap, and addressing key application facets. This minimizes ambiguities, streamlines development, and ensures alignment with the initial vision.

2. **Reflection Phase:** The reflection phase is crucial for refining and clarifying specifications between the AI assistant and the human collaborator. This phase preempts potential pitfalls and ambiguities, ensuring a smoother development process.

3. **Infrastructure Implementation Phase:** The Infrastructure This phase establishes the foundational environment for the application's development, testing, and deployment, hosted either on-premises or cloud-based. It involves setting up the development environment and installing the operating system as the base layer for subsequent software components.

4. **Application Implementation Phase:** In this critical stage, AI systematically develops software components, starting with foundational utility classes like database managers. This phase includes code generation and creating Unified Modeling Language (UML) diagrams to illustrate the system's architecture and component interactions. Prompts should guide the AI to detail both backend and frontend class structures. Effective prompts include:

   a. "Please generate an initial design for each class, outlining responsibilities, attributes, and methods, including descriptions for front-end and back-end classes."

   b. Request the AI to generate the DDL for the database: "Please generate the DDL for Database and tables."

   c. Configure the database and begin generating each class: "Please generate the complete Database Utility class with all appropriate methods."

   d. For DAO structures, prompt the AI to generate the DAO class, followed by the Data Model class: "Please generate the complete X class with all getter and setter methods, and include a main method for testing."

   e. Conduct unit testing for each class. If errors occur, copy the error message and prompt: "Logic Error: [error message]." ChatGPT will typically generate a fix.

   f. Repeat steps d and e for each class until all are developed.

5. **Debugging:** Each developed class undergoes immediate testing, ensuring incremental development. This cyclical implementation and debugging process ensures a robust, error-free product. After all classes are tested, perform an integration test.

6. **Post-Deployment Review:** A rigorous post-development review is essential to ensure the application meets requirements and adheres to industry best practices. This review focuses on security measures to identify and mitigate vulnerabilities, crucial in today's evolving cyber threat landscape. Prompts such as "Please review the Student Class Planner

Application code for security issues, specifically SQL injection, and recommend modifications" are useful at this stage.

7. **Deployment Phase:** In the deployment phase, AI-crafted prompts streamline the process, ensuring efficient transition to the target environment. The AI provides precise, comprehensive instructions tailored to specific deployment needs.

8. **Maintenance:** Maintenance is a critical and enduring phase. The Software Orchestration process, with its adaptability and precision, facilitates swift and efficient system modifications.

In summary, Software Orchestration introduces a paradigm shift, leveraging AI's capabilities to enhance each phase of the software development lifecycle. From the meticulous of prompts to the iterative process of implementation and debugging, Software Orchestration epitomizes the fusion of human expertise and AI's computational prowess.

3 Methods

In Section 3, we present an experimental methodology to evaluate the Software Orchestration process, focusing on the granularity of the initial prompt and its impact on development. Our research uses a two-dimensional framework: the first dimension assesses initial prompt granularity at three levels, and the second dimension examines application complexity in three classes—Class A (1-5 classes) Task Manager, Class B (5-10 classes) Club Membership, and Class C (10-20 classes) Online Book Store. This structure results in nine experimental scenarios, each providing distinct insights. The study aims to understand how the orchestration process handles varying complexity and interdependencies, and whether it scales effectively. It explores the optimal detail level in initial prompts to balance human input with AI accuracy and efficiency, crucial for establishing best practices in AI-assisted software engineering. Additionally, it analyzes error propagation across different complexities and prompt details, offering insights into potential vulnerabilities and mitigation strategies in the orchestration process.

4 Results

This section analyzes the results from experiments evaluating Software Orchestration in application development across three classes (A, B, and C) of varying complexity and size. These classifications provided a structured framework to assess ChatGPT's performance in orchestrating development tasks, including

Experiment	Reflection Count	App Size Max LOC	Initial Prompt Detail	Syntax & Logic Errors	Prompts	Classes	Duration Minutes	Time (min) /Prompt	Time (min) /Class	Prompts /Class	Actual Lines of Code
Task Manager Application											
A1	6	500	Low	7	16	3	50	3.13	16.67	5.33	371
A2	10	500	Medium	3	11	4	38	3.45	9.50	2.75	377
A3	8	500	High	2	10	4	35	3.50	8.75	2.50	419
Club Membership Planner											
B1	8	1000	Low	6	25	10	76	3.04	7.60	2.50	925
B2	8	1000	Medium	7	28	8	81	2.89	10.13	3.50	591
B3	6	1000	High	3	20	7	70	3.50	10.00	2.86	799
Online Book Store Application											
C1	10	1500	Low	16	22	13	95	4.32	7.31	1.69	714
C2	16	1500	Medium	7	24	10	79	3.29	7.90	2.40	923
C3	12	1500	High	12	14	13	80	5.71	6.15	1.08	1570

Table 1: Experiment Results

response accuracy, error handling, and overall efficiency. Key metrics such as reflection question count, syntax and logic errors, prompt counts, class generation, development duration, lines of code, and time efficiency per prompt and class were recorded and analyzed.

In analyzing the data (Table 1) [3], several key trends emerge. Class A applications, being smaller and less complex, showed a decrease in development time as prompt detail increased from low (A1) to high (A3). The low-detail experiment (A1) had more syntax and logic errors compared to the high-detail experiment (A3), suggesting that more detailed initial prompts reduce errors. In contrast, Class B and C applications displayed a different pattern. Medium-detail prompts (B2 and C2) had the highest syntax and logic errors, indicating that for more complex applications, medium-detail prompts might introduce complexities that are not immediately evident.

The time efficiency metrics varied across classes, with no clear trend, highlighting the unique challenges of each application size and prompt detail level. However, high-detail initial prompts generally lead to more efficient development, potentially reducing time for clarification and error correction.

Class C applications required more development time overall but were more efficient on a per-class basis, especially in the C1 (low detail) experiment. This highlights the complex dynamics of software orchestration and the influence of application size and prompt detail.
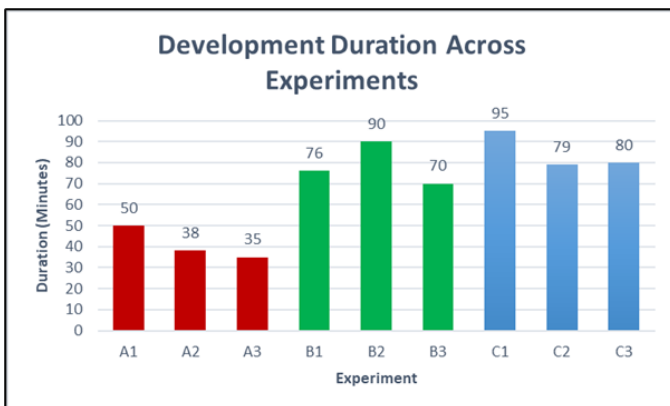
Figure 1: Development Duration (Minutes)
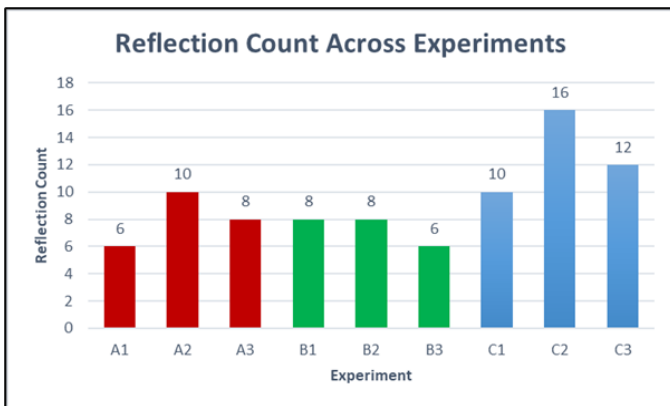
Figure 2: Reflection Count

The reflection count (Figure 2) decreases with increased detail in the initial prompt for Class B and Class C. Specifically, in both Class B and Class C, the experiments with the highest level of initial prompt detail (B3 and C3) recorded the lowest reflection counts within their respective classes. This suggests that more detailed prompts may lead to a clearer understanding of the task at hand, thus reducing the need for further clarifications or additional guidance, as indicated by the lower number of reflection questions. However, this trend does not apply to Class A. Class A applications, being smaller and less complex, might inherently involve fewer uncertainties or ambiguities, making the effect

of prompt detail less pronounced.

5   Discussion

The empirical data from our experiments with Software Orchestration, integrating ChatGPT into the software development process, yields several insights.

1. **Detail in Prompts:** Detailed design inputs consistently led to higher quality applications, underscoring the need for well-articulated concepts for effective AI collaboration.
2. **Medium-Detail Prompts:** These presented challenges, often resulting in gaps during implementation, indicating a critical threshold in the granularity of information needed for optimal AI performance.
3. **Efficiency:** The ability to deliver a working prototype application in (C3) 80 minutes, albeit small 1570 source lines of code (SLOC) is impressive. For comparison purposes, using the Basic COCOMO model [1] for an "organic" project (characterized by small, experienced teams with flexible requirements), the effort to develop 1 KLOC (1000 lines of code) is estimated at 2.4 person-months, with a development time of approximately 3.49 months. This estimate suggests that a single developer might take about 3.5 months to complete 1000 lines of code (or 5.5 months for 1570 SLOC), though actual times can vary due to factors like code complexity, developer experience, and project management practices.
4. **ChatGPT's Performance:** The model managed smaller applications well but faced limitations with extensive or complex designs, suggesting constraints in its short-term memory or processing capabilities.
5. **Low-Detail Specifications:** ChatGPT tended to make assumptions, sometimes missing specific requirements and compromising completeness.
6. **Error Handling:** ChatGPT efficiently identified and proposed multiple solutions for syntax and logic errors, showcasing its utility in iterative debugging.
7. **Prompt Clarity:** Clear, direct prompts resulted in more complete outputs, while vague prompts led to placeholders like TODOs, emphasizing the need for unambiguous communication.
8. **Structured Development:** A structured process, including an initial prompt followed by ChatGPT's reflective questions and software architect clarifications, proved beneficial, especially for Class A and B prompts.
9. **GUI Design:** Detailed UI descriptions were crucial for effective GUI application development, highlighting the need for explicit guidance.

6 Background and Literature Review

The use of artificial intelligence (AI) in software development has been explored in various studies. AI-assisted software development has the potential to significantly improve productivity. Moroz [7] highlights the benefits of AI in software development, particularly in the form of neural network programmer's assistant Copilot. This AI tool can help with correct command formulation, copyrighting, and safety issues, ultimately enhancing productivity. O'Connor [8] and Mohammadian [6] both highlight the potential of AI in enhancing software project management, with O'Connor specifically focusing on distributed multi-platform projects. Ebbah [2] further discusses the application of AI techniques in solving software engineering problems, emphasizing the complexity of the software development process. Kulkarni [5] emphasizes the integration of AI activities in software development processes, which can lead to improved product quality. However, Trendowicz [10] notes that while AI tools can enhance productivity, the capabilities of developers and the tools and methods they use still play a significant role.

7 Summary and Conclusion

Our findings suggest that Software Orchestration, integrating ChatGPT in software development, holds significant promise by expediting the development process and delivering functional applications. The correlation between prompt detail and application quality implies that more specific prompts enhance productivity. Additionally, ChatGPT effectively supports software design documentation, generating PlantUML and various UML diagrams, highlighting its potential beyond code generation. However, the approach has limitations. The complexities of software orchestration revealed areas for improvement. Future advancements in Large Language Models are anticipated to enhance the efficacy and scope of Software Orchestration. This study illustrates the current capabilities of AI-assisted software development and opens avenues for further research and development in this field.

References

[1] Barry Boehm. *Software Engineering Economics.* Prentice-Hall, 1981. ISBN: 0-13-822122-7.

[2] Jonathan Ebbah. "Deploying Artificial Intelligence Techniques In Software Engineering". In: *American Journal of Undergraduate Research* (2002). DOI: 10.33697/AJUR.2002.006. URL: https://doi.org/10.33697/AJUR.2002.006.

[3] Joseph Elarde, Barry Bruster, and Mir Hasan. *Software Orchestration: A Paradigm for Software Development and Security Assessment using ChatGPT resources.* `https://github.com/jves1/SoftwareOrchestration01.` Accessed: 2024-05-17. 2024.

[4] GeeksforGeeks. *Software Engineering | Classical Waterfall Model.* `https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/`. Accessed: 2023-10-07. 2023.

[5] Rajesh Kulkarni and P. Padmanabham. "Integration of artificial intelligence activities in software development processes and measuring effectiveness of integration". In: *IET Software* (2017). DOI: `10.1049/iet-sen.2016.0095`. URL: `https://doi.org/10.1049/iet-sen.2016.0095`.

[6] Masoud Mohammadian. "Innovative Applications of Artificial Intelligence Techniques in Software Engineering". In: *Proceedings of the AIAI Conference.* 2010. DOI: `10.1007/978-3-642-16239-8_3`. URL: `https://doi.org/10.1007/978-3-642-16239-8_3`.

[7] Ekaterina A. Moroz et al. "The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement". In: *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus).* 2022. DOI: `10.1109/ElConRus54750.2022.9755659`. URL: `https://doi.org/10.1109/ElConRus54750.2022.9755659`.

[8] Rory V. O'Connor and J. Jenkins. "Using agents for distributed software project management". In: *Proceedings of the IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99).* 1999. DOI: `10.1109/ENABL.1999.805175`. URL: `https://doi.org/10.1109/ENABL.1999.805175`.

[9] OpenAI. *ChatGPT: Optimizing Language Models for Dialogue.* `https://openai.com/chatgpt`. Accessed: 2023-10-07. 2023.

[10] Adam Trendowicz and Jürgen Münch. "Factors Influencing Software Development Productivity - State-of-the-Art and Industrial Experiences". In: *Advances in Computers, Vol. 77.* Elsevier, 2009, pp. 185–241. DOI: `10.1016/S0065-2458(09)01206-6`. URL: `https://doi.org/10.1016/S0065-2458(09)01206-6`.

A Capstone Experience for Undergraduates in High Performance Computing With *MPI* and *OpenMP**

Andrew J. Pounds
Department of Computer Science, Mercer University
Macon, GA, 31207
`pounds_aj@mercer.edu`

Abstract

The assignment described herein is intended as a capstone experience for students in a scientific or parallel programming course in which they have utilized shared memory symmetric multiprocessing using *OpenMP* and been briefly exposed to distributed parallel processing using *MPI*. Both *OpenMP* and *MPI* are established, well-documented, libraries for doing parallel computing and students traditionally find *OpenMP* to be very easy to implement. *MPI* is more difficult, but is necessary if one wants to communicate between compute nodes. To fully take advantage of a computing cluster, one can combine these technologies - utilizing *OpenMP* to do shared memory programming on each system while using *MPI* to do the inter-system communication. Herein we describe the hardware and environment needed for the project, the base code provided to students, the problems that students faced, the methods of performance analysis that students used, and common errors made over numerous offerings of the course.

1 Introduction

Over the past few decades high performance computing (HPC) has been slowly moving into the undergraduate curriculum [1, 2] with some success. However,

---

teaching HPC at primarily undergraduate institutions can still be challenging due to many factors [3] including a lack of computational resources or appropriate material targeting undergraduates. Another indicator that HPC is becoming more important in the undergraduate curriculum is that parallel computing is now listed as one of the ABET curriculum requirements [4] for computer science. As such programs may introduce rudimentary parallel programming as part of the introductory courses. However, to address many of the concepts needed by developers of high performance computing code, additional training is needed. For example, if one is interested in developing or maintaining codes for scientific modeling or data analysis, students will need to have a working knowledge of numerical analysis and linear algebra and be conversant in one of the regarded "high performance computing" languages (C/C++/Fortran). Thankfully developing shared memory parallel programs in these languages has been made much simpler via the implementation of the *OpenMP* library. With *OpenMP* one can literally add a few statements around computationally intensive code segments to add parallelism.

OpenMP [5] is designed for parallelism on a single system with multiple cores utilizing shared memory. The Message Passing Interface, *MPI* [6], can do the same thing, but *MPI* is capable of communication between systems connected via a network. Because one has to explicitly handle sending and receiving data, *MPI* is significantly more difficult than *OpenMP* for students to implement, but it is necessary if one wants to take advantage of multiple computers in a cluster environment. A hybrid method of parallel programming that offers the best of both worlds is to write code that breaks the problem across several systems using *MPI* and then uses the shared memory parallel programming model of *OpenMP* on each system to further enhance performance. This concept is not new and has been widely implemented in HPC codes and is an active area of research in the HPC community. [7–11]

The project described herein is designed for computer science and engineering students that have completed a semester of numerical analysis and are currently enrolled in a course in which they have learned about common code optimization techniques, the use of HPC hardware, compiler based optimization, and *OpenMP* shared memory parallel programming. They have been introduced to the fundamentals of *MPI* and given working matrix multiplication code with which to experiment and test. Matrix multiplication was chosen for the performance testing because it is well understood, relatively easy to code, and provides enough computational load that students can see their performance enhancements across multiple processors and nodes. In addition, students can easily check their work for accuracy by utilizing one of the system linear algebra libraries. The code students received contained no *OpenMP* directives.

While this submission could be considered by some to fit into the category of "Nifty Assignments", the fact that it is really designed for advanced students, requires some familiarity with numerical analysis, and depends on students having some background in concepts related to parallel computing moves it away from that classification. In addition, completing the project can require running hundreds of test trials that are then analyzed to make date driven decision on how to proceed.

2   Environment

While developing HPC code traditionally involves using supercomputers and systems located at national labs, times have changed. Many of the tools that were once proprietary are now freely available and parallel computing can be accomplished at just about any undergraduate institution with an appropriately outfitted computer lab. [12] All of the software and analysis tools mentioned herein are either free or open source. Cloud computing has also significantly lowered the barrier to teaching HPC courses and this project could easily be completed on cloud resources.

2.1   Hardware and OS

Any computer lab with networked multicore computers could be used to complete this project. In the present work 20 computers each with 10 core processors capable of dual hyper-threading were used. Each processor had a three level cache structure and was connected via the bus to 16 GB of RAM. All computers were connected via gigabit Ethernet to a high speed switch. Each computer was running a 64bit Linux OS (CentOS 7).

2.2   Compilers

The compilers from the Linux distribution are satisfactory for this assignment, and can be used with the corresponding *OpenMP* and *OpenMPI* libraries that can also be downloaded from the Linux repositories. The system compilers, however, can sometimes lag significantly behind the current compilers that have features that the students want to use. For that reason, in the present work, the newest Gnu Compiler Collection was downloaded and compiled from scratch with support for POSIX threads to allow for shared memory parallelism.

2.3   Parallel Tools

The Gnu compiler suite also includes the *OpenMP* library and development files. To avoid any issues with the newer compiler and the older *OpenMPI*, a

newer version of the *OpenMPI* source was downloaded and compiled using the newer compiler.

2.4   Analysis

For the analysis of the results, students were provided with some Perl scripts that would parse their result files and order the data for plotting. The *gnuplot* program was used to build the 3D surface and contour graphs of performance as a function of nodes and processors per node.

2.5   Batch Environment

While not strictly necessary, a batch system to control the flow of jobs onto the computing cluster is extremely helpful. Depending on the size of the class there could be thousands of benchmarking jobs waiting to execute. The use of a batch scheduler can also help prevent students from over-subscribing nodes and thus negatively affecting their benchmarking results. For the present work the *Torque* resource manager and the *Maui* job scheduler were used.

3   Methods

Starting Point: Prior to this exercise student had thoroughly optimized and parallelized with *OpenMP* their own linear algebra library [13] including matrix multiplication. Matrix multiplication is a dense matrix technique that requires $2N^3$ floating point operations and students had previously used this information to build performance graphs of megaflops as a function of number of cores. For this exercise students were provided with a working matrix multiplication program in C that utilized *MPI* and they had to modify it to only time the matrix multiply function using the MPI_Wtime function. [14] All three matrices, in their entirety, were sent to each of the nodes. All of the message passing was done via synchronous MPI_Send and MPI_Recv transactions. In a class demonstration is was easily verified that the performance increased with the number of nodes – but that this performance increase was short lived.

Proof of Concept: With performance benchmark data in hand for a single core on each system, students were asked to implement *OpenMP* pragmas in the code to see if they could improve the performance. For the vast majority of the students this was a trivial task and thus demonstrated that the code could be made faster by implementing *OpenMP*.

Test Cases: To ensure that the results were correct after their code modifications, students were instructed to build three different test cases. No

points were awarded if a program gave results outside of predetermined tolerances.

1. Utilize three pre-built fixed-dimension matrices, $\hat{A}$, $\hat{B}$, and $\hat{C} = \hat{A}\cdot\hat{B}$ determined with a linear algebra library, and then take the difference of the matrix product $\hat{A}\cdot\hat{B}$ and the pre-computed matrix $\hat{C}$. Sum up the diagonal. It should be zero.

2. Build a random matrix $\hat{A}$ and use a linear algebra library to compute the inverse of the matrix. With $\hat{B} = \hat{A}^{-1}$, compute the product $\hat{C} = \hat{A}\cdot\hat{B}$ and sum up the diagonal of $\hat{C}$. It should be equal to the dimension of $\hat{A}$.

3. Construct a matrix $\hat{B}$ that is row major with the first element starting at 1 and each subsequent element being equal to the previous element plus 1. For example a 4x4 matrix would have the form.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Then define matrix $\hat{A}$ as the identity matrix. The product of $\hat{A}$ and $\hat{B}$ should then produce a copy of $\hat{B}$.

Each of these test cases can be used to test different things. For example both test cases 2 and 3 will work if either the matrix $\hat{A}$ or its transpose is used; this can be useful in identifying if matrices are being sent and received correctly. The ordered matrix in the third case can also be helpful in determining if buffers are being filled in the correct order. Finally, the matrices from the first test case will only give correct answers if everything is working correctly.

Reducing Message Passing Overhead: While the code provided to students worked, it was extremely inefficient because it sent all of the matrices to each of the processes. In actuality, only matrix $\hat{A}$ needs to be sent to each of the processes; the appropriate columns of matrix $\hat{B}$ can be split among the nodes. This can result in a significant savings in network transmission overhead. The drawback is that not sending the entire matrix forces the students to think about how the matrices are stored in memory and how to pack, send, and receive the correct matrix elements needed by each process. In addition, each worker process only needs to create arrays big enough to receive the matrix elements sent to it and the developer needs to think how to modify the matrix multiplication process

in the worker code to correctly perform the multiplication task with the smaller set of data.

In the original code provided to the students the test matrices were being built in the main method before *MPI* was initialized. This meant that all of the matrices were being built in their entirety on each of the processes. To reduce the workload the construction of the matrices were moved to only the master process. Moving this build process exclusively to the master meant that the only matrices found in the worker processes were sent to the worker by the master. Doing so, however, exposed numerous errors generally associated with message passing parallelism. Properly moving data between systems had to be dealt with before students could move forward.

Implementing *OpenMP*: The provided code had numerous loops that were ready for shared memory parallelization using *OpenMP*. Almost all of the loops, however, contained data dependencies that hindered parallelization. Students could still parallelize the loops using *OpenMP* pragmas, but if they did not address the data dependencies they never achieved optimal scalability and maximum performance.

Collective Communications: Since the entirety of matrix $\hat{A}$ has to be sent to each of the nodes, student could take advantage of *MPI* collective communications. Using collective communications, matrix $\hat{A}$ is *broadcast* to the network and each of the worker nodes receives it simultaneously.

4 Results

Before doing any tests with *OpenMP*, students were asked to benchmark the two pieces of *MPI* code provided to them that utilized a single thread on each node. One code copies all of the matrices to the worker processes and the second piece of code copies only the portions of matrix B needed to accomplish the work assigned to that node. The codes were benchmarked with matrices of dimension 2000 and 4000. The results are shown in Fig. 1. With this data in hand, students were asked to think about how they would implement *OpenMP* to get even greater performance. The obvious option was to write *OpenMP* pragmas around the triply nested matrix multiplication kernels in the master and worker processes. Almost every student immediately added the pragma shown in Listing 1.. In Listing 1, `startIndex` and `stopIndex` are the starting and stopping column indices of matrix $\hat{B}$ assigned to the process. `*(subA)` contains the entirety of $\hat{A}$ and `*(subB)` ONLY contains the needed columns of $\hat{B}$. The use of `bufIndex` makes it very easy to put the data correctly into the `*(subC)` memory location - but it has to be remembered that `*(subC)` is a
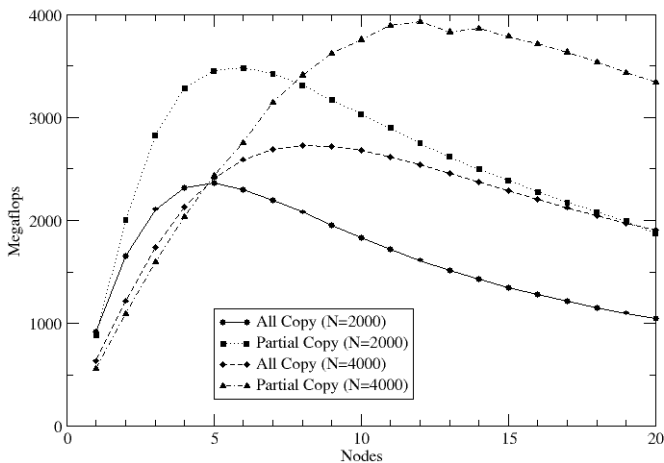
Figure 1: Performance results for multiplying two NxN matrices as a function of number of nodes. Based on the results it is obvious that using partial copies provides a significant performance enhancement and that performance improves with larger matrix dimensions.

Listing 1: A Naive attempt to parallelize an existing loop with OpenMP. The serialized `bufIndex` inhibits full parallelization.

```
#pragma omp parallel for default(shared) private(i,j,k)

  for ( int i=0; i<NDIM; i++)
    for (int j=startIndex; j<=stopIndex; j++) {
      for (int k=0; k<NDIM; k++)
        *(subC+bufIndex) += *(subA+i*NDIM+k) *
            *(subB+((j-*(start+rank))*NDIM)+k);
      bufIndex++;
    }
```

Listing 2: An improved attempt to parallelize an existing loop with OpenMP. The formally serialized bufIndex can now be calculated independently in each thread as a function of the index variables.

```
#pragma omp parallel for default(shared)
                                private(ii,jj,kk,bufIndex)
  for ( ii=0; ii<NDIM; ii++)
    for ( jj=startIndex; jj<=stopIndex; jj++) {
      bufIndex = (jj-startIndex)*NDIM+ii;
      for ( kk=0; kk<NDIM; kk++)
        *(subC+bufIndex)  += *(subA+ii*NDIM+kk) *
                *(subB+(jj-startIndex)*NDIM+kk);
```

linearized array the same size as *(subB) and will be passed back to the main program.

The benchmark results for the Listing 1 are shown in Fig. 2a. While the results do improve with the number of threads, it is far less than optimal because the bufIndex is serialized and limits parallelization. The code has to be re-written to remove this dependency and compute the value of bufIndex as a function of the indexing variables. One such way to do this is shown in Listing 2. Any student that chose to make these changes got performance results that resembled those shown in Fig. 2b. What should be noticed in

(a) Using code from Listing 1. (b) Using code from Listing 2.
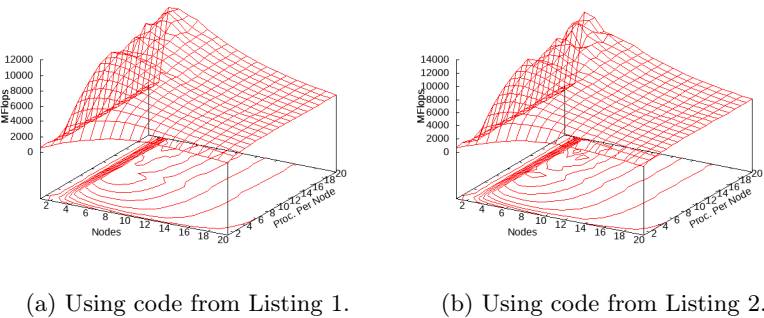
Figure 2: Performance results for 4000x4000 matrices as a function of number of nodes and processors per node. Overall performance improved by more than 2 gigaflops and scaled to more processor cores.

Fig. 2 is that while the code is performing roughly 20% better on a smaller number of nodes, it did not scale well across multiple nodes. Astute students
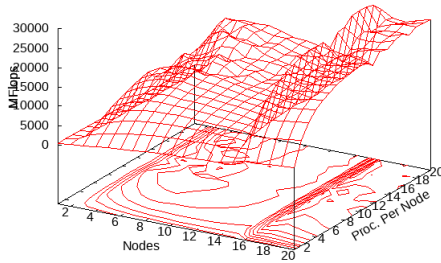
Figure 3: Performance results for 8000x8000 matrices as a function of number of nodes and processors per node after modifying the code to eliminate loop dependencies, parallelize the packing and unpacking of matrix buffers, and doing all-to-all communication. The overall performance of the code doubled from prior attempts and scaled to a maximum at 17 nodes with 18 cores per node. The jump in performance after 15 nodes is believed to be due to the level 3 cache once again being unsaturated since.

opened two terminals, one on the node running the master task and one on a node running the worker task, and ran *top*. They noticed that, after the workers completed the parallelized matrix multiplication kernel, the code on the master would never use more than two cores and the worker node would drop back down to serial, single-core performance. Students surmised that this was due to the packing and unpacking of the array buffers before and after the MPI_Send/MPI_Recv. They thus proceeded to rewrite all of those sections of code, which also required removing data dependencies, in parallel using *OpenMP*. This resulted in modest, but improved, scalability across nodes. Some students spent a significant amount of time in the *MPI* documentation and tried to use all-to-all MPI_Bcast collective communication methods to minimize the time for sending matrix $\hat{A}$ to each of the processes. The results for those that implemented in correctly is show in Fig. 3.

5 Discussion and Conclusions

Even though writing parallel *OpenMP* code is relatively easy for students, writing distributed parallel *MPI* code is a skill they must master if they want to code across computing clusters. In this exercise students were eager to enhance the *MPI* code with their *OpenMP* calls. However, to make significant

improvements they had to not only learn how *MPI* works, but they also had to dive deeper back into their own *OpenMP* code to understand what aspects were hindering parallelization and remove data dependencies. This is a common theme in HPC coding - fixing one piece of code often breaks something else. Each time a new piece of hardware is introduced one has to often learn how to write code for it in order to build better-performing HPC applications. Only through a careful analysis of the benchmarking results can one tell if their code modifications are making a difference. This concept was reiterated numerous times while students completed this exercise.

Normally a paper of this type would include a breakdown of learning objectives and how they were assessed in the implementation of the assignment over a number of years to different cohorts. Since this assignment was used in a course offered biennially to a small number of students, such a statistical analysis is unjustified. Some qualitative observations on learning objectives, however, are noteworthy. In the course of using this assignment four times in the past seven years, only two students out of a total of thirty-three took the time time to carefully investigate why the code was not scaling across multiple nodes and produced a solution that satisfactorily solved the scalability problem. Every student, however, had to focus on data dependencies, parallel and serial regions, memory management, data locality, thread safety, message passing, and synchronization. These are the hallmarks of a parallel computing curriculum; this assignment demonstrates the need to consider all of these areas when looking for performance enhancing solutions and rewards students that take time to do so. In all of the instances where this assignment was used students worked tirelessly to try to improve their code. They saw that their diligence in reading documentation, critically analyzing code, using graphical methods to visualize and analyze performance, trying new and different coding strategies, and taking a data-guided approach to code modification, gave a marked improvement that was commensurate with their level of work.

HPC coding can be very rewarding when it is done thoughtfully and analytically. This assignment reinforces this type of code development by methodically walking students through several levels of parallelization in a systematic and data-driven manner to produced demonstrably faster code. The assignment also stimulated a healthy sense of competition among the students that greatly enhanced the level of engagement.

References

[1] Andrew Fitz Gibbon et al. "Teaching high performance computing to undergraduate faculty and undergraduate students". In: *Proceedings of the 2010 TeraGrid Conference*. TG '10. Pittsburgh, Pennsylvania: Asso-

ciation for Computing Machinery, 2010. ISBN: 9781605588186. DOI: 10.1145/1838574.1838581. URL: https://doi.org/10.1145/1838574.1838581.

[2] B. Neelima and Jiajia Li. "Introducing high performance computing concepts into engineering undergraduate curriculum: a success story". In: *Proceedings of the Workshop on Education for High-Performance Computing*. EduHPC '15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450339612. DOI: 10.1145/2831425.2831430. URL: https://doi.org/10.1145/2831425.2831430.

[3] Jordan Hebert et al. "High-performance computing in undergraduate education at primarily undergraduate institutions in Wisconsin: Progress, challenges, and opportunities". In: *Education and Information Technologies* (2024), pp. 1–25.

[4] ABET. *Criteria for Accrediting Computing Programs, 2023 – 2024 | ABET*. 2023. URL: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2023-2024 (visited on 05/30/2024).

[5] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*. MIT press, 2007.

[6] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*. Nov. 2023. URL: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf.

[7] Qinnan Qiu et al. "An efficient hybrid MPI/OpenMP parallelization of the asynchronous ADMM algorithm". In: *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 2021, pp. 563–570. DOI: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00083.

[8] Vladimir Mironov et al. "An efficient MPI/OpenMP parallelization of the Hartree-Fock method for the second generation of Intel® Xeon Phi™ processor". In: *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2017, pp. 1–12.

[9] Pedro Valero-Lara et al. "MPI+OpenMP tasking scalability for multi-morphology simulations of the human brain". In: *Parallel Computing* 84 (2019), pp. 50–61. ISSN: 0167-8191.

[10] Jérôme Richard et al. "Fine-Grained MPI+OpenMP Plasma Simulations: Communication Overlap with Dependent Tasks". In: *Euro-Par 2019: Parallel Processing*. Ed. by Ramin Yahyapour. Springer International Publishing, 2019, pp. 419–433. ISBN: 978-3-030-29400-7.

[11] J. F. Stanton et al. *CFOUR, Coupled-Cluster techniques for Computational Chemistry, a quantum-chemical program package.*

[12] Andrew J. Pounds, Rajeev Nalluri, and Bennie L. Coleman. "The development of a Tri-Use cluster for general computer education, high performance computing education, and computationally intensive research". In: *Proceedings of the 43nd Annual Southeast Regional Conference, 2005, Kennesaw, Georgia, USA, March 18-20, 2005, Volume 1*. Ed. by Mário Guimarães. ACM, 2005, pp. 345–348. DOI: 10.1145/1167350.1167446.

[13] Andrew J. Pounds. "The babyblas - an extended project for introducing undergraduates to the concepts of high performance and parallel scientific computing". In: *J. Comput. Sci. Coll.* 28.2 (Dec. 2012), pp. 153–159. ISSN: 1937-4771.

[14] Lawrence Livermore National Labs. *Workshop example.* 2022. URL: https://hpc-tutorials.llnl.gov/mpi/exercise%5C_1/ (visited on 04/30/2022).

Impact of a Virtual Peer Mentoring Program (VPM) on Student Retention and Performance*

Hyesung Park, Sonal Dekhane, Wei Jin, Lorraine Jonassen
[1]Department of Information Technology
Georgia Gwinnett College
Lawrenceville, GA 30043
hpark7@ggc.edu, sdekhane@ggc.edu, wjin@ggc.edu, ljonassen@ggc.edu

Abstract

This study examines the impact of the Virtual Peer Mentoring (VPM) program on Information Technology (IT) students' retention and academic performance at an open access, majority minority-serving institution. The VPM program was designed during COVID-19 for IT majors as a small group peer mentoring initiative where junior or senior mentors met weekly with groups of up to six freshmen or sophomore mentees virtually. The program incorporated best practices from research studies and was adapted to meet the needs of the target student population. Data collected from Fall 2021 to Spring 2023 student participation indicated higher retention rates for students attending the program. GPA differences between completers and non-completers showed clear benefits from attending the program, especially for students from underrepresented and underserved groups. The findings highlight the importance of integrating peer mentoring with freshmen orientation and existing coursework to address recruitment and retention challenges, particularly for underrepresented and economically disadvantaged students.

1   Introduction

COVID-19 had a significant impact on higher education, impacting enrollments and learning modalities. At the authors' public college, which is an open access and majority minority-serving institution, small class sizes, student engagement and face-to-face learning were central ideas aligned with the institution's mission. Before COVID-19 only one fully online course was offered, and a couple of courses were hybrid. Emphasis on out of class faculty-student engagement had resulted in an active campus community with seven IT-focused student clubs, student participation in regional competitions, hackathons, scholarships, etc. Given the demographics of about 50% Pell grant recipient students and about 25% first-generation students, building this community had taken significant time and effort. The sudden switch to online classes and COVID-19's impact on student lives affected the department's enrollment and failure and withdrawal rates. COVID-19 also impacted faculty-student engagement with many out-of-class activities completely canceled. For a few activities, such as the Peer Supplemental Instruction program (PSI) that moved online, student engagement was almost non-existent. Hence, the Virtual Peer Mentoring (VPM) program was designed to support incoming freshmen and rising sophomores to help them navigate college during this difficult time. A pilot study was conducted based on one year's worth of data to investigate the program's impact on participants' sense of belonging. The positive findings of this study [self-reference] resulted in this program being offered beyond one year. This paper investigates the following research questions based on data collected from two years of implementing the VPM program:

1. What is the impact of the VPM program on student retention at the college and in the major?
2. How do GPA outcomes differ between completers and non-completers of the VPM program?
3. What are the benefits of the VPM program for underrepresented and economically disadvantaged students at the college.
4. How does participation in the VPM program affect Pell Grant recipients' academic performance over time?
5. How do students from different demographic groups respond to the VPM program in terms of retention and GPA?

2   Literature Review

The term mentor has been defined in the literature using various terms, such as a coach, a tutor, and an experienced, trusted advisor who offers personalized guidance and support to others [3, 8, 4]. Mentoring programs today appear in

many formats, including formal, informal, diverse, cultural, collaborative multilevel, and electronic. Formal mentoring is a planned and structured format, whereas informal is a spontaneous interaction between mentor and mentee. Diverse, cultural, collaborative, multilevel, and group mentoring focuses on equality across genders, races, ethnicity, and levels. Electronic mentoring, also known as e-mentoring, incorporates online platforms. Each of these mentoring styles can exist in one-to-many, many-to-one, many-to-many formats, or a combination of all three [8, 2]. Lastly, the current trend is peer mentoring, known for its reciprocal and collaborative approach where two groups hold each other accountable as they reach shared goals [5]. Peer mentoring has been recognized for its potential to enhance student retention and academic achievement, especially among underrepresented and economically disadvantaged groups. According to authors [4], "Self-efficacy is built as peer mentors who serve as role models provide academic and emotional support to peers."

Literature reviews aligned explicitly with this study offer a diversified understanding of peer mentoring and its many concepts. To start, peer mentoring in higher learning is not new; it dates back decades. For example, in 1986, researchers [9] conducted a pilot program in which they recruited upper-level students at the City University of New York (CUNY) to mentor first-year high-school students. The role of the mentors was to provide mentees with academic and social guidance. A comparison between the mentored and non-mentored groups revealed a significant increase in mentees' attitudes toward school, yet it showed no significant increase in attendance and grade point average. In support of future mentoring endeavors, the authors recommended the integration of coursework into the process along with collaboration between faculty, mentors, and mentees.

To address enrollment challenges, Mount Holyoke, a women's liberal arts college, designed a Megas and Gigas Educate (MaGE) program to prepare students to become peer mentors [7]. Topics covered in their course included: effective learning, motivation techniques, self-efficacy, and growth mindset strategies emphasizing diversity, inclusion, and social identity. Following completion of the training, peer mentors were hired to work with students in CS1 or CS2 classes. The goal was to create an inclusive and sustainable program to address the needs of underrepresented and minority students, which it did. Following completion of the course, mentees reported they felt like they belonged in the computer science community and their peer mentors were knowledgeable, approachable, and creative, which helped them to succeed. Ultimately, Mount Holyoke's enrollment doubled; they went from a 36-seat lecture to two 36-seat lectures after the first year.

Building upon their Megas and Gigas Educate (MaGE) program, researchers [6] at Mount Holyoke presented a training curriculum focused "on inclusion as

a key tool for creating a welcoming environment that fosters a community of learning". In collaboration, the computer science faculty designed the curriculum with a psychology and education colleague. Specific teaching modules included instruction on effective learning on various topics, such as motivation, self-efficacy, growth mindset, and technical skills. Following a mentoring semester, mentors and mentees reported significantly increasing their abilities and self-confidence. The researchers also noticed that the acquired skills were transferable to future employment opportunities.

Researchers [1], in a 2016 study, examined factors that could influence first-year college students' intent to persist in their academic journey. A total of 172 students from diverse backgrounds completed surveys administered at the beginning and end of their first semester. The survey questions focused on academic variables (GPAs and ACT scores), self-efficacy beliefs, participation in learning communities, mentorship, and socioeconomic status. A standard multiple regression analysis revealed that college self-efficacy beliefs and mentorship could significantly predict students' intent; however, they cannot be used to predict academic performance related to grades.

For this study, the authors put the institution and the student's needs at the forefront and used the MaGE platform as a guide to design the VPM program. Evidence-based best practices from different peer mentoring programs were adapted to meet the students' needs. A detailed program description of VPM and results of the pilot study are described in [self-reference]. The following sections of the paper discuss the mentoring sessions' topic details, data collection and analysis, and discussion and conclusions.

3   Program Description

The VPM program was designed as a small group peer mentoring program, where each peer mentor, a junior or senior IT major met with a group of up to six mentees who were freshmen and/or rising sophomore IT majors online once a week for a semester.

A kick-off event at the start of the semester and an end of semester celebration event were used to bring everyone together and build a sense of community among students. This included the mentors, mentees and faculty advisors. The kick-off event consisted of various activities, such as an icebreaker, pre-surveys, welcome by the Dean, introduction of the program, its goals and expectations, and a dedicated time for networking with peers. An IT faculty member also spoke at each kick-off event and presented their undergraduate research project(s) along with their undergraduate research students to introduce students to undergraduate research opportunities.

The end of semester celebration was relatively informal and was used as

an opportunity to celebrate mentor and mentee accomplishments, administer post-surveys, and share concrete steps to stay connected with the community. A graduate student panel was conducted to increase students' awareness about graduate school opportunities, possibilities and benefits.

The weekly sessions were designed to be interactive and peer mentors were trained to engage mentees in the conversation. Peer mentors were trained to acknowledge that mentees brought life experiences and knowledge with them and could share information from which everyone could benefit. Mentees were also required to engage in follow-up activities that required the application of information received at mentoring sessions. Additionally, sessions also included group study. A typical semester-long peer mentoring program covered the following topics:

1. The first session was focused on growth mindset to help students develop a mindset for learning, growth and perseverance.
2. A session focused on planning for a successful semester and building relationships with faculty and peers was included early in the semester to provide students with the tools and information to prepare for a successful semester.
3. Sessions introducing students to campus resources for academic support, such as the Peer Supplemental Instruction (PSI) program for gateway STEM courses, academic enhancement center that offers tutoring services and various workshops, technology support, etc. were included and representatives of these programs were invited as guest speakers for the campus.
4. To improve students' sense of belonging and engagement in the department, sessions included information about IT student clubs, service-learning opportunities, regional competitions and hackathon opportunities, etc. Students from various student clubs and other student success initiatives were invited as guest speakers for these sessions.
5. A session focused on scholarship opportunities based on need, discipline or academic performance introduced students to an array of programs that offered financial scholarships or aid to continue or complete their education. This also included programs that offered travel grants for students to attend conferences. Programs at the college and those external were presented to the students. Scholarship and grant recipients were invited as guest speakers.
6. Peer mentors also conducted a session on presentation skills that focused on creating and delivering effective presentations. Mentees were required to create and deliver short presentations as a follow-up to this session.
7. To help students navigate college, sessions focused on describing the difference between various IT concentrations offered at the college, and un-

derstanding and using academic maps and mentoring milestones were offered.

8. Sessions focusing on professional development opportunities, such as internships, undergraduate research opportunities, and resources offered by the career advising center on campus were offered. Guest speakers from the career advising center and students who had participated in undergraduate research and internships were invited to be guest speakers for these sessions.

9. Around mid-terms and closer to the end of the semester, sessions focused more on group study, problem-solving strategies and upcoming engagement opportunities such as summer boot camp, programming refresher session, preparing for an IT company field trip, etc.

4 Data Collection and Analysis

In this paper the authors present data collected from VPM implementation from Fall 2021-Spring 2023 semesters. The program was not offered during the summer semesters. A total of 123 students applied and were accepted into VPM during these semesters. Of these 123 students 58 students (47.15%) did not attend any sessions at all. Of the remaining 53%, 23 (35.38%) students attended less than 50% or 8 sessions. 42 or 64.61% students attended 50% or more sessions during the semester. Overall, more women than men sought to participate in the program, 67 (54.44%) vs 56. This is a higher representation as compared to women's representation in the IT major, which ranges between 20%-25% at the college. About 57 (46%) of the students who sought to participate in VPM were Black/African American. Their numbers in the college and in the IT major range around 33%-36%. Data provided by the college's Institutional Research (IR) office in Spring 2024 shows that of the students who were accepted into the program but did not attend any sessions, 46.55% are no longer registered in classes at the college. Whereas 30.95% of those who attended half or more sessions are no longer registered at the college. The data also shows that almost 60% of students attending 50% or more sessions have progressed in their academic journeys, thereby seeing a status change. Instead, only 37.93% of the students who did not attend any sessions have progressed in their academic journeys. The data shows similar change in students switching majors. Whereas 17.24% of those who did not attend any sessions switched their majors out of IT, only 4.76% of those who attended 50% or more sessions did so. Additionally, 7.14% of those students who attended 50% or more sessions switched their majors to IT upon completing the program.

The following figures compare the GPA of VPM participants based on their completion or non-completion of the program. Completers and non-completers

were identified using attendance records. Completers are students who attended at least 8 sessions a semester. GPA data was provided by the IR office for semesters fall 2021 – fall 2023.
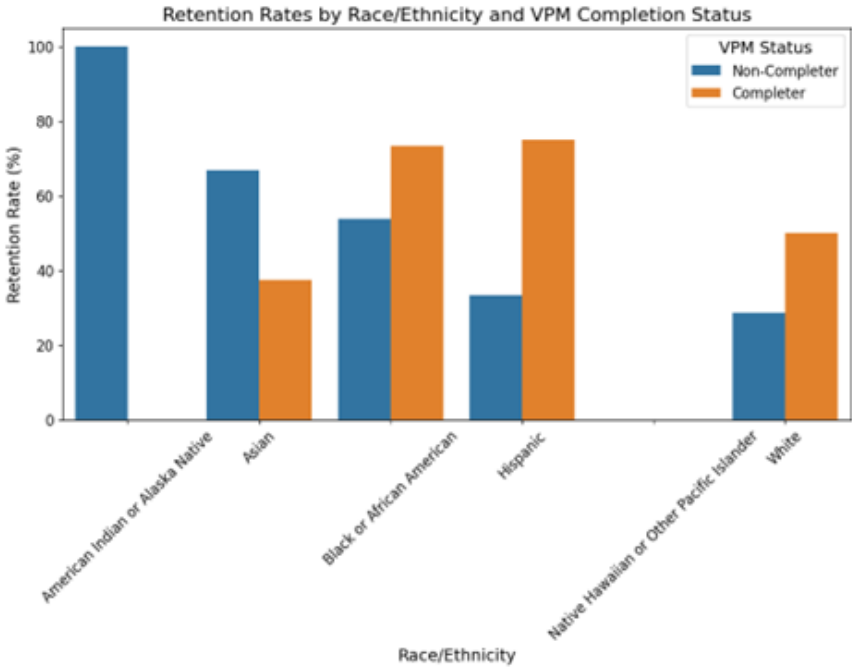
Figure 1: Retention by race/ethnicity among completers vs non-completers

Figure 1 shows that Black/African American, Hispanic and White completers retained at the college at a higher rate than non-completers.

Figure 2 shows that both men and women who completed the program performed better than those in their group who didn't.

Figure 3 shows that Black/African American completers had much better GPAs than non-completers, as did White completers. Hispanic completers also had better GPAs than non-completers. A Mann-Whitney U test (nonparametric test) was conducted because group (incomplete vs. complete) data are not normally distributed, violating the assumption of normality for both groups. Therefore, an ANOVA test was not appropriate to run the significant test. Table 1 below shows the results of the Mann-Whitney U test where all the p-values are less than 0.05, therefore, suggesting that there is significant difference in GPA between students who completed the VPM program and those who did not, across all terms.
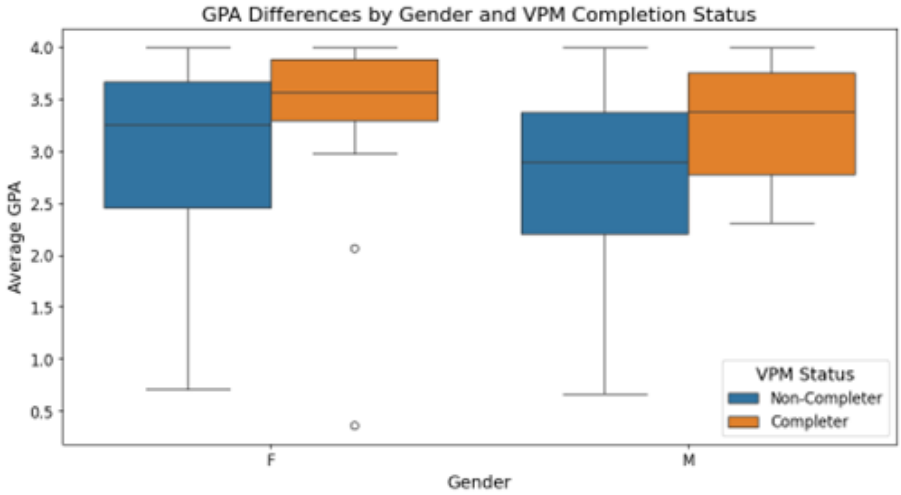
Figure 2: GPA differences by gender and attendance
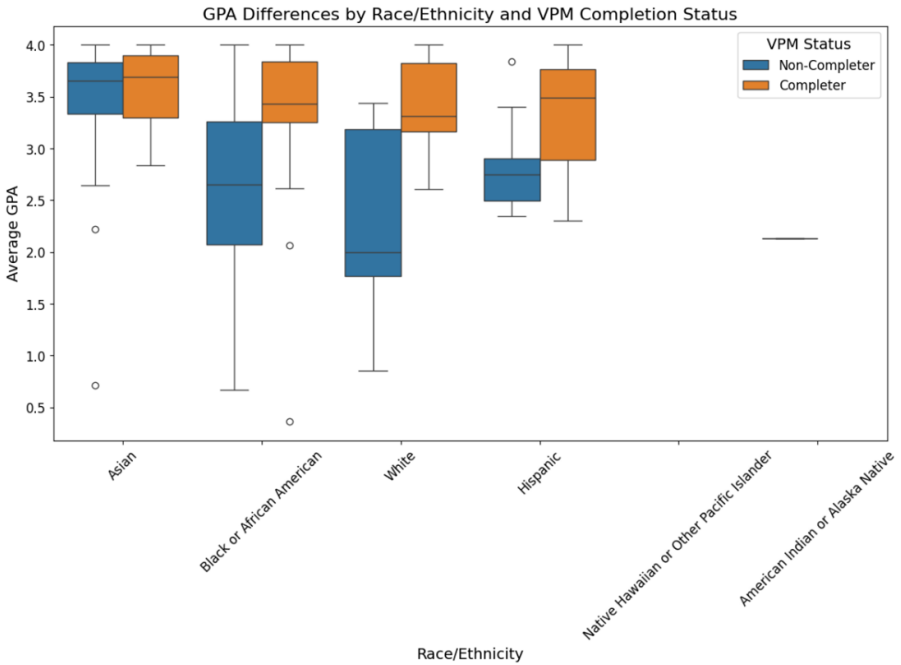
Figure 3: GPA differences by race/ethnicity and attendance

Table 1: Mann-Whitney U Test

	Fa21	Sp22	Fa22	Sp23	Fa23
Mann-Whitney U	302.50	468.50	837.00	944.50	975.50
Wilcoxon W	1248.50	2179.50	3538.00	3870.50	3901.50
Z	-3.177	-3.355	-3.074	-2.824	-2.634
Asymp. Sig. (2-tailed)	.001	$<$.001	.002	.005	.008
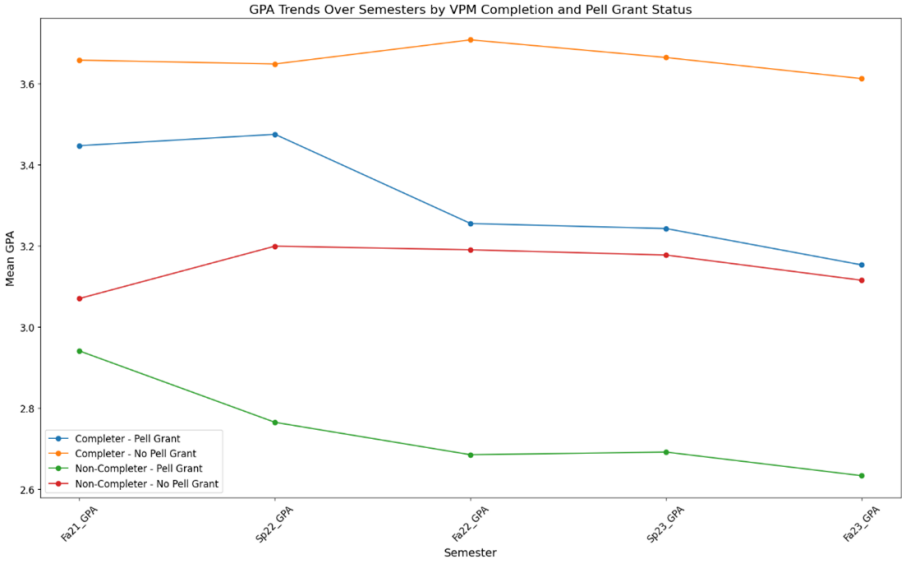
Figure 4: GPA trends over semesters

About 67% of the VPM participants were Pell recipients. Figure 4 shows that among non-completers (lower two lines in the chart), both Pell recipients and non-recipients had fairly close average GPAs in Fall 2021. However, for those without a Pell grant, the GPA increased slightly and remained mostly stable. In contrast, for Pell recipients, the GPA decreased and continued to decrease over time. Similarly, among completers (top two lines in the chart) Pell grant recipients started at a lower average GPA in Fall 2021, saw an increase the following semester, but then continued to decrease over time. Non-Pell recipients among the same group started at a higher overall GPA, which increased and then decreased, still remaining much higher than Pell recipient completers. The GPA difference between Pell recipient completers vs. non-completers is significant. While a decrease in GPA over time among all students could be explained by the increasingly challenging coursework in the major, the

differences in GPA among Pell recipients and non-recipients are significant. Non-completer Pell recipients have the lowest average GPA and a decreasing trend.

5 Discussion and Conclusion

The data from two years of VPM implementation during and immediately following COVID-19 at a public, open-access, majority minority-serving institution shows that women and Black/African American students were more likely to seek peer mentoring opportunities. Their representation in VPM was higher than their representation in the IT major. Students who completed the program retained at the college at a higher rate than those who were accepted in the program but did not attend any sessions. Completers also retained at a higher rate in the IT major as compared to those who did not attend any sessions. Retention rates of Black/African American and Hispanic students who completed the program were significantly higher than those who didn't, establishing the need for such programs to help broaden participation in computing. GPA differences among completers and non-completers of different genders and race/ethnicity also show clear benefits of VPM for students from underrepresented groups. Lastly, the data shows that Pell grant recipient completers benefitted from VPM participation before their GPA decreased, indicating a need for continued support to sustain the increased GPA trend among this group. Pell grant recipient non-completers had the least GPA and a decreasing trend, indicating a need to increase efforts for engaging them in VPM. While completers came into VPM with a higher GPA than non-completers, it is important to note that about 30% of completers were not retained in college and more than 4% switched majors. While these numbers may seem small, their existence indicates that the risk of losing students, especially those from underrepresented and economically disadvantaged backgrounds exists. The impact of VPM and its need for students from diverse groups (underrepresented in STEM, low economic background, first generation, etc.) cannot be dismissed. Ultimately, recruitment and retention in the program remain a challenge. Competing priorities at home and work for students could be significant factors, along with lack of awareness about the value of programs like this, especially for incoming freshmen. For this reason, the authors believe that further integration with freshmen orientation and existing coursework could help address the recruitment and retention challenges. Hence, this emerges as a top priority for future direction. Such integration could also help address challenges associated with resources at smaller institutions as student support and success becomes part of existing responsibilities and curriculum, instead of an additional burden for faculty.

Acknowledgment

References

[1] Stefanie T Baier, Barry S Markman, and Francesca M Pernice-Duca. "Intent to persist in college freshmen: The role of self-efficacy and mentorship". In: *Journal of College Student Development* 57.5 (2016), pp. 614–619.

[2] Nuria Gisbert-Trejo et al. "Mentoring programs implementation: differences between group and individual mentoring". In: *Development and Learning in Organizations: An International Journal* 36.4 (2022), pp. 1–4.

[3] Thomas Landefeld and Thomas Landefeld. *Mentors and mentoring.* Springer, 2010.

[4] Carol A Mullen and Cindy C Klimaitis. "Defining mentoring: a literature review of issues, types, and applications". In: *Annals of the New York Academy of Sciences* 1483.1 (2021), pp. 19–35.

[5] Judy O'Neil and Victoria J Marsick. "Peer mentoring and action learning". In: *Adult Learning* 20.1-2 (2009), pp. 19–24.

[6] Heather Pon-Barry et al. "A flexible curriculum for promoting inclusion through peer mentorship". In: *Proceedings of the 50th ACM technical symposium on computer science education.* 2019, pp. 1116–1122.

[7] Heather Pon-Barry et al. "Addressing the CS capacity challenge by improving undergraduate peer mentoring". In: *ACM Inroads* 8.3 (2017), pp. 43–47.

[8] Heidrun Stoeger et al. "Nine years of online mentoring for secondary school girls in STEM: An empirical comparison of three mentoring formats". In: *Annals of the New York Academy of Sciences* 1483.1 (2021), pp. 153–173.

[9] Susan B Turkel and Theodore Abramson. "Peer tutoring and mentoring as a drop-out prevention strategy". In: *The Clearing House* 60.2 (1986), pp. 68–71.

Basic Document Retrieval with Retrieval-Augmented Generation through UIUC.chat*

William Paul Kelly[1]
[1]University of North Carolina Wilmington
Wilmington, North Carolina, U.S.A.
`WPaulKelly271828@gmail.com`

Abstract

When given a prompt containing a particular fact or statement, UIUC.chat was able to use Retrieval-Augmented Generation (RAG) to identify a provided document as containing the given statement. This has obvious applications, but is not especially novel by itself. Future efforts will focus on comparing two documents through UIUC.chat to find similarities and discrepancies in statements made. This may be useful towards a longer-term goal of creating a Large Language Model (LLM) that can collate large numbers of documents to produce a broad overview of a given field.

1 Initial Purpose and Findings

The following was obtained through UIUC.chat, an open-source, web-based frontend preprocessor which uses ChatGPT to generate responses. One feature of UIUC.chat is to use RAG to compare prompts given by the user to various user-provided documents. UIUC.chat then adds relevant portions of those documents to the original prompt before sending it to ChatGPT for a response.
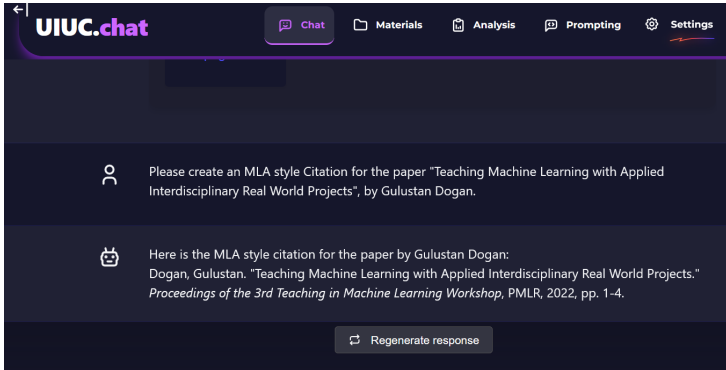
Figure 1: UIUC.chat produces a basic citation.
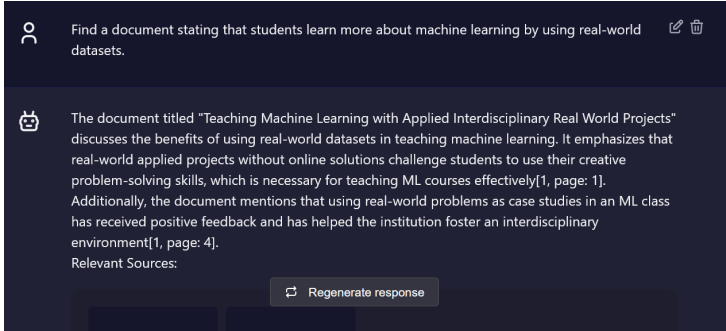
Figure 2: UIUC.chat retreives a document.

The initial goal was to find a way of generating MLA-style research citations by passing a document and a prompt to an LLM. This project had been suggested some time ago, and since then basic LLM functionality has improved to where UIUC.chat was able to accomplish most of this goal with minimal work, shown in Figure 1.

UIUC.chat was able to produce MLA-style citations with minimal, if any, adjustment required. Further work could be undertaken to make the citations produced more consistent and complete, but this was difficult to justify, as the end result would only offer marginal benefit over simpler, preexisting citation generating tools. Instead, work was done to determine if UIUC.chat's RAG functionality could retrieve a document when given a prompt containing a statement or fact written in that document, before then creating a research citation for that document. This was also achieved with minimal work required.
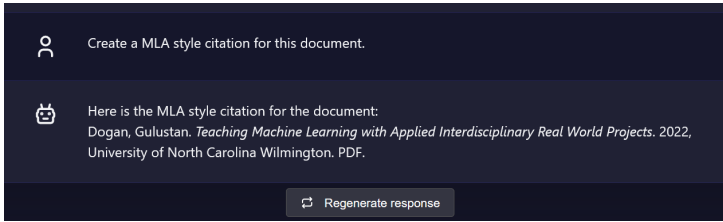
Figure 3: UIUC.chat creates a citation for the document.

Although the results obtained above have obvious utility - for example, to assist and enhance research - they alone are not substantial enough to represent a novel use of LLMs. The ability to answer questions about and generate research citations for given documents could serve as parts of a useful tool for collating and producing research papers. However, this would only act as a marginal extension of the core functionality of the LLMs being used, and in fact such tools already exist, such as Google's NotebookLM. More ambitious goals are necessary to create a novel application for LLMs.

2   Related Works

Moving forward, we would like to focus instead on comparing two documents to one another to find similarities and discrepancies in the statements made by both. Some initial research has been undertaken to assess overall feasibility, common approaches, and potential applications.

In their work on detecting contradictions in formal engineering requirements, Alexander Elenga Gärtner and Dietmar Göhlich have established different classifications of contradiction[1]. These classifications may be useful to our own work, particularly if contradictions between our documents can be shown to fall within their system, as they established for their own working domain. They also note the necessity of using a combination of formal logic and LLMs to identify contradictions in natural language.

Adian Liusie, Potsawee Manakul, and Mark Gales showed the feasibility of using pairwise comparison to assess NLG output, rather than a scoring system, helping to establish the utility of document comparison[2]. Some of their findings suggest that pairwise comparison might be surprisingly practical in comparing large quantities of documents, a task which might intuitively appear more suited to a scoring system; this helps support a potential applications of document comparison introduced later in this paper.

Stefan Trapp and Joachim Warschat have showcased the use of prompt engineering alongside GPT-4 to identify contradictions within the inventive

concepts of patents[3]. Although the definitions and context are much more formalized than in some of our cases, the underlying techniques of comparing statements remains relevant to our work, and the prompt engineering applied will likely also prove useful.

3   Next Steps

Even barring techniques shown above, rudimentary document comparison can likely be achieved by a modest extension of what we've already accomplished, via making more prompts regarding basic content checks of the documents involved. Despite how feasible this goal is, it may act as a valuable step towards creating truly novel technologies which accomplish things that haven't been accomplished yet.

3.1   Scope

Rather than be limited to a particular domain of inputs, it would be ideal to attain more generalized techniques for document comparison. These techniques might act as a rough guideline for comparing any two objects that can be modelled as documents; these guidelines could then be tailored to better fit the particular type of documents being compared. Two document types of interest are research papers and nutritional information. Research papers can be compared to identify points that the two papers might agree or disagree on, as well as areas not covered by either paper. Nutritional information of food products can be compared to identify common ingredients, nutrients not covered adequately, and other useful information.

These were selected for two reasons. First, there are practical applications to the automated comparison and analysis, discussed later in this paper. Second - and more significantly, in the short term - these two types of documents are heavily opposite to one another in terms of reliance on natural language versus strict formatting rules. Research papers, while ultimately based in fact, can be largely arbitrary in terms of how an author chooses to express their ideas, and thus will likely need to be parsed heavily by an LLM to be analysed. In contrast, nutritional information of a registered food product generally follows a strict template, to the point where certain logical and mathematical constructs might be adequate to fully represent them given proper context. In studying the comparison of both of these types of documents, not only can we make progress towards two different useful applications, but information can also be gained about necessary differences in approach between comparing simple and complex documents. This may help to create better guidelines on when LLM comparison is necessary, how prompts should be engineered to account for different documents, and other similar problems.

In considering research papers, this project does not make special cases for misinformation, or instances where an author deliberately seeks to mislead the reader. We will specifically consider documents that are well-regarded, and which are generally believed to have been written in good faith. This scope of consideration can be changed as necessary.

3.2   Methodology

At worst, comparing any two documents should be as straightforward as listing the statements made by both documents and then comparing all statements by brute force. Computationally this would not be ideal, but there may be ways of simplifying the number of comparisons required, such as through detecting keywords between statements before comparing them fully. ChatGPT may even have mechanisms to simplify this process which can be accessed through good prompt engineering. Existence of such mechanisms is made more plausible by the number of projects seeking to compare documents for purposes such as requirements engineering and legal proceedings: Because there is already desire to compare documents in this fashion, it is also more likely that current tools have already begun to accommodate this desire.

4   Long-Term Goals

LLMs, in their current state, are unlikely to consistently outperform a well-trained human at comparing documents. Because of this, it would be ideal to apply automated document comparison in a fashion that humans might struggle with, such as comparing large quantities of documents with one another. Both of our chosen types of test document might yield useful applications when reasoned about in large numbers.

4.1   Research Papers

Even outside of large-scale comparison, a general model for comparing two academic papers with LLMs might be directly useful as a research aid in several ways. If a researcher sees that two papers disagree on outcomes, this model might help them more quickly identify which sections of the papers are most relevant. If a researcher plans to expand on findings from a specific paper, they might compare that paper to other papers to identify which findings are generally agreed with, and which ones are more often opposed. This might then encourage the researcher to pursue their work to address one of these points: If something is agreed on, the researcher might choose to take that point as a given and use it as part of something more ambitious. If something is disagreed on, they might choose to resolve the ambiguity with a more thorough

investigation of that particular aspect of the work. Finally, they might even compare a rough draft of a paper they've written to existing papers, to confirm that they've acknowledged the perspectives that they intended to, and possibly also to identify anything they might have overlooked.

As shown above, there could be a great deal of utility in comparing a paper to many other papers, not just for the benefit of one project, but for many projects within the field. There may be potential for an AI which takes in large numbers of papers and gradually analyses and collates them one-by-one, to ultimately identify the overall shape of findings and disagreements in a field. This might take the form of a publically available web portal, displaying some kind of star chart of significant papers or talking points, which users could then click to learn more about, as well as see related papers and topics. Such a system could help both experienced researchers to see how the full breadth and context of their field is developing, as well as helping newcomers to understand what major talking points are well-established and which ones yet need to be further clarified. This might lead users to more easily identify strong points to build on or weak points to resolve within the overall body of research in a field. There may also be potential to identify other characteristics of a collection of research in this way, such as identifying common topics and related fields.

4.2   Nutritional Information

Comparing large quantities of nutritional information might also have significant applications, particularly when taken alongside geolocational data. Comparing a large sample of food products in an area might help indicate the presence of food deserts, or heavy reliance on crops that might be vulnerable to conditions such as drought or extreme temperature. Inclusion of additional data, such as the sale price at a particular location, might reveal and create hard evidence about relations between nutritional content and price. Information on using LLMs to compare less rigidly-defined aspects of nutritional data might be used alongside more conventional approaches as part of a larger system. This system, with appropriate inputs, might be able to reason about a large number of important topics, such as food systems susceptible to climate change; regions at risk of producing diet-related illnesses; potential nutrition-based explanations for local epidemics; and logistics assistance in connecting producers, processors, and wholesalers.

4.3   Closing

As with smaller-scale comparison, future research in creating these information nexuses might be useful in an almost unforeseeable number of industries. LLM document comparison might create extremely powerful tools for reasoning

about large quantities of data, particularly in cases such as published research papers or food products, where the entire population is known and parameters can be determined directly rather than statistically.

References

[1] Alexander Elenga Gärtner and Dietmar Göhlich. "Automated requirement contradiction detection through formal logic and LLMs". In: *Springer* 31.49 (2024). DOI: https://doi.org/10.1007/s10515-024-00452-x.

[2] Adian Liusie, Potsawee Manakul, and Mark Gales. "LLM Comparative Assessment: Zero-shot NLG Evaluation through Pairwise Comparisons using Large Language Models". In: *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Yvette Graham and Matthew Purver. St. Julian's, Malta: Association for Computational Linguistics, Mar. 2024, pp. 139–151. URL: https://aclanthology.org/2024.eacl-long.8.

[3] Stefan Trapp and Joachim Warschat. *LLM-based Extraction of Contradictions from Patents*. 2024. arXiv: 2403.14258 [cs.CL]. URL: https://arxiv.org/abs/2403.14258.

Starting a Civic Engagement Capstone: An Experience Report*

Mark Hills and James B. Fenwick, Jr.
Department of Computer Science
Appalachian State University
Boone, NC
`{hillsma,fenwickjb}@appstate.edu`

Abstract

In this paper, we reflect on the first semester of a Computer Science capstone course focused on civic engagement projects. This was offered as an alternate section of an existing capstone, with multiple student teams working with a local non-profit. We describe how the student work was structured during the semester, as well as the activities leading up to the semester and occurring immediately after the semester. We also reflect on what worked well, and discuss potential changes in future semesters. We believe this will be helpful to other faculty that are either starting a new capstone course, or modifying an existing capstone course, that will engage community partners with student teams.

1 Introduction

Starting in Spring 2024, the Computer Science Department at Appalachian State University introduced a special section of its capstone course focused on *civic engagement* projects—projects with a client (in our case, a local non-profit) selected from the surrounding community. Civic engagement is an informal approach to experiential, service learning. Service learning has its philosophical and pedagogical roots in the work of Dewey [5], Freire [6], and

---

Kolb [8]. A key tenet is abandoning the idea of education being a "banking process" that fills students with knowledge and facts. Instead, education is viewed as a reciprocal relationship where teachers also learn and learners also teach. This approach can support deep learning as students work up Bloom's taxonomy [1] synthesizing classroom knowledge in practical and local settings. Students benefit in multiple and varied ways, including academic knowledge and skills, interpersonal communication skills, and increased self confidence [13].

In this paper, we reflect on our initial experience in adding a civic engagement section of our capstone course. Many students enter college with idealistic ambitions of changing the world. Sometimes these ambitions can become lost in the curricular routine of assignments and short-lived, academic projects. Capstone courses offer students an opportunity to revisit those early ambitions with more skill and knowledge, while adding a service-learning component affords students an easy entry to working on a fulfilling project with a positive, immediate, and direct impact on an observable community organization. We start in Section 2 by describing the preexisting capstone course. Section 3 then provides details about the civic engagement section of the capstone, including what was done before the semester began, student and faculty tasks during the semester, and student reflections on their work once the semester concluded. Section 4 summarizes the results of an informal survey sent to a CS education mailing list to solicit information about similar capstones in other programs, while Section 5 highlights some existing research on capstone courses in computer science and software engineering (including some highlighted by responses to our survey). Section 6 concludes by reflecting on what worked well and what modifications may be made in future semesters to overcome difficulties encountered during the Spring 2024 semester. Our hope is that this paper will be helpful to faculty in other programs that are considering starting such a capstone or that have recently started such a capstone.

2   Background

The current capstone course, CS 4800, is generally taken by undergraduate students in their final year in the program. The catalog description of the course is: "The senior capstone project provides the student an independent and collaborative software development experience with a significant project. The course introduces aspects of project management, requirements analysis, and the software lifecycle, but will primarily be concerned with the practical integration of core theories, practices, and ethics of the discipline. Writing and speaking communication skills are reinforced." Students enrolling in the course must have senior standing, and must have already taken the Software Engineering and Data Structures courses, the latter of which is also a gateway

to many of our upper-level electives. Typical enrollment is around 20 students per section. The course meets for 3 hours once per week. During this time, students generally give status updates, present the current status of their work, and work on their projects. Although students are expected to also work on the projects outside of class, the class session provides dedicated project time each week, which can be especially helpful if they are stuck and need assistance from the instructor. Having the full team together in person also helps re-center the team as students often have challenges in aligning their out-of-class schedules.

Students in a typical section of CS 4800 work on personal, self-selected projects. Most students work on individual projects, but occasionally students with shared interests work on teams of 2 or (more rarely) 3 students. Early in the term, students propose a project idea which, if approved, they work on over the course of the semester. The technologies are up to the students, and vary based on student interests and the type of project. Core courses in the program use Java, C/C++ and SQL, with other common languages including Python, JavaScript, Swift, and Kotlin (depending on which other courses students take). While many students use these languages, and the techniques used in earlier courses, some students use the capstone as an opportunity to learn a new language or platform. Several checkpoints, including a final report and a final project presentation, occur throughout the semester. Common examples of projects from past semesters include games, personal assistants, and utilities, with console, GUI, web, and mobile applications all being common choices. The typical capstone project starts and ends during the academic term. In some cases students may begin with an existing personal project, and the capstone would continue building on what they had already developed.

3   The Civic Engagement Capstone

In this section, we provide details about the Spring 2024 civic engagement capstone section. We look at activities before the start of the semester, including the inception of the civic engagement section; describe the activities of the capstone during the semester; and look at the results of the semester, including client and student feedback.

3.1   Before the Semester

During the Fall 2023 semester, we were approached by an employee of the non-profit that we worked with in the capstone. They had received a grant to expand their services and wanted technology assistance to develop a client portal. Shortly after this, the university Center for Excellence in Teaching and Learning for Student Success sent out a Request for Proposals, focused on integrating civic engagement into a course. Inspired by this RFP, we decided to add

a special section of our capstone course focused on civic engagement projects. Although the immediate goal was to teach a special section in Spring 2024, the longer-term goal was to continue this work into future semesters, providing a mechanism for students to get involved in team projects with community partners that often have a lifetime of more than a single semester. In contrast to the projects described in Section 2, these projects would not be created by the students, but would instead be based on the needs of our community partners, and would (except in unusual circumstances) all be team projects. We applied for the RFP and were awarded funds to help cover small stipends for the faculty involved, to purchase technology books for the students, and to travel to speak with other faculty working on similar capstone courses.

3.2   During the Semester

The capstone course met a total of 14 times over the course of the Spring 2024 semester. Faculty met with two possible clients before the start of the semester: a community partner (the local nonprofit mentioned above, which is the focus of this experience report) and a startup. During week 1, the instructor presented initial details about the clients and their needs, then worked with the students to develop questions for each client for a planned client visit. This occurred during week 2, with each client visiting the course, presenting further details about their needs, and answering questions from the authors and the students. After the clients departed, the instructor and the students decomposed the overall work for each client into multiple projects: five for the community partner, and two for the startup. The projects for the nonprofit were an API backend server, a web client, and three mobile clients: an Android client, an iOS client, and a cross-platform client (added to explore the design space of possible mobile solutions). Students then ranked their interest in each project in a survey form and were assigned to teams based on their responses.

Week 3 then started the actual work on the projects. 17 students were divided into 5 teams for the nonprofit: 3 students on each mobile project, and 4 on the API server and web client projects. Each student project was organized according to Scrum principles, with two week sprints, self-directed student teams, and the instructor acting as the Scrum Master, helping students follow best practices. During the first week, each team spend class time to get organized for the rest of the project. The instructor created teams for each student team on GitHub. Each team then created one or more GitHub repositories for code collaboration, and created GitHub project boards to track their work. They also started planning the actual work of the project, creating initial issues to represent team tasks and making initial decisions on technology (e.g., the technology stack they would use) and teamwork (e.g., collaboration workflows for effectively using GitHub). This was the start of Sprint 1.

Each week, before class, each team submitted a single status report. These reports were submitted by creating a document on a shared folder, with the goal of providing a history for future semesters. Each class meeting started with a Scrum standup meeting, where each team could describe what they had accomplished over the past week and what issues were preventing them from making progress. Class meetings then ended with details of what the team planned to accomplish over the next week. During class time, teams could work together on their projects and meet with the clients. Every other week, the teams would present the current state of their project, with the goal of quickly moving towards a minimal viable product that would provide a basis for future work. Meetings with the community partner were either held over Zoom or were asynchronous, with the teams coordinating to email collections of questions instead of each emailing the client individually. Meetings with the startup were held either in-person or over Zoom. Student teams communicated using a Discord server created by the instructor or using other preferred communication mediums (e.g., text messages, other Discord servers, email).

In the week before the final class meeting, the student teams focused on ensuring the project was in a good state to be picked up by the next teams in the fall. The semester then ended with the submission of a project report (built over the course of the semester, with the instructor giving feedback at multiple checkpoints) and final project presentations by all teams, which were attended by the clients. Each student team also met individually with the instructor for a more detailed project demonstration and in-depth discussion. In general, students made good progress on individual projects, but work on linking the projects (e.g., the website to the backend API server) has mainly been deferred to Fall 2024.

3.3 After the Semester

At the end of the semester, a student survey with the following questions was sent out to all students in the civic engagement section of the capstone:

Q1 What teamwork skills did you improve, or learn, during the project?

Q2 How effective do you believe your team was? (1 to 5 scale, 1 is dysfunctional, 2 is ineffective but functional, 3 is sufficient, 4 is effective, 5 is highly effective)

Q3 What do you believe could have been done better to help your team work together more effectively?

Q4 What technical skills did you improve, or learn, during the project?

Q5 What do you believe could have been done better to help you learn new technical skills during the project?

Q6 How much communication was there with the other teams? (1 to 5 scale, 1 is Not Enough, 3 is Just Right, 5 is Too Much)

Q7 How much communication was there with the client? (Same scale as Q6)

Q8 Did communication with the client come at the right times? (1 to 5 scale, 1 is at the wrong times, 5 is at just the right times)

Q9 How did you communicate with the client? (Email, Text, Telephone, Discord, Zoom, In Person, or Other, multiple answers allowed)

Q10 What could be done in the future to improve communication with external clients? What worked well, and what could be done differently?

Q11 Feel free to share any other thoughts about your experience working on your project.

Students could only respond once to the survey, but student emails were not revealed as part of the collected data so responses were anonymous.

Of the 23 students in the class, 7 responded to the survey. Note that some of these responses may have been from the students on the projects with the startup—while still useful, they do not bear directly on the experience of the students working with the community partner. The students gave their teams an average effectiveness of 3.43 (question 2). Looking at communication, respondents gave average ratings of 2.43 for question 6 (communication between teams, with 3 being "just right"), 1.71 for question 7 (communication with the client, same scale), and 2.57 for question 8. All students selected "email" as one communication option, with all but one also selecting Zoom. One common theme was the importance of communication with the client, and the need for more regular, scheduled times to meet. Comments included "I think it would be very helpful if the client worked closer with the groups", "Meeting with them in person more often", and "I think having more face to face or zoom calls would be helpful". We discuss this further in Section 6.

We also met with the community partner for a debriefing session. In general, they were excited about the progress the students made during the semester. They were especially happy with the coordination of the teams around visual elements, such as the use of the same fonts and color schemes that they use in their materials. They had several ideas for additional features, and committed to staying involved during the Fall 2024 semester.

4 An Informal Capstone Survey

To identify other programs that offer project-based courses with external clients, especially those that have projects that extend across semesters, we sent a survey to the ACM SIGCSE (Special Interest Group in Computer Science Education) mailing list. Note that the survey was intentionally kept quite short and informal—our goal was to make it easy to respond, giving us a chance to gather some initial information which could then be enriched with future conversations. This survey included the following questions:

Q1 Does your undergraduate degree require students to take a capstone course near the end of their time in your program? Note that this may be called "Capstone", but could also be a project-based course under another name.

Q2 If you answered "Yes" to Question 1: Do you allow your students to work on projects with community partners and other outside individuals and/or organizations?

Q3 If you answered "Yes" to Question 2: Do you have projects that last for more than a single term/semester?

Q4 Would you be willing to either have a short conversation over Zoom with us or answer additional questions about your capstone course?

Q5 Do you have any additional thoughts to share about capstone courses, working with external partners/on external projects, or multi-term projects?

We received a total of 16 responses to the survey. Of these, 14 stated that their programs offer some kind of capstone experience for their students, with 1 additional program offering a course that, while not a capstone, has students work on a significant software project. Ten of the programs with a capstone course require students to take this course. All 15 of the programs with a capstone or capstone-like course allow engagement with community partners, while 13 of these programs have projects which can last more than one semester, some by design (e.g., a two-semester capstone sequence), some by choice of the student (e.g., a student can retake the course for additional credit and can continue the same project). Seven of the responses stated that a project may be worked on by different student teams over time.

While we cannot generalize these results to programs outside of those of the respondents, the results have given us more information about how the responding programs run similar capstone courses. We plan to carry out a more rigorous study in the future, across a broader collection of programs, that will hopefully yield more detailed information.

5   Related Work

As described by Coyle et al. [3] and by Oakes et al. [12], Engineering Projects in Community Service (EPICS) is a model for integrating community engagement throughout the curriculum. Students participate in projects that span multiple years, with project teams made up of students from multiple disciplines, including Computer Science. Started at Purdue [16], EPICS has expanded to multiple other universities, including Princeton, Ohio State University, and Arizona State University [17]. Linos et al. [10] describe adapting the EPICS model to a new Software Engineering program in a liberal arts setting, including a discussion of ongoing challenges around topics such as the continuity of projects and effective project management. One significant difference between the work on EPICS and the work we describe above is that we are just focused on the capstone course at this point, so students are all senior-level students that are just taking the course for a semester.

Davis and Rebelsky [4] describe the design of a software design course which includes multi-semester projects with community partners. Kurkovsky et al. [9] introduced a service-learning framework named Scaffolded Projects for the Social Good, or SPSG. This uses a studio-based approach for student projects with external partners, and is similarly run during development using agile techniques with two-week sprints. For continuity, students can work on the same projects across an undergraduate software engineering and a senior project or capstone class, and spend the last week of the semester focused on knowledge transfer activities. Braught et al. [2] describe multiple models for introducing H/FOSS (Free and Open Source, potentially with Humanitarian goals) projects into the curriculum, with models varying on the organization of the capstone course and related courses, such as courses in software engineering. All of these models appear to have goals similar to ours, but with potentially different course sequences, so we plan to follow up with the authors.

Dugan [7] conducted a survey of existing literature on undergraduate capstone courses in Computer Science. This survey organized the literature around both course and project goals, including topics such as which software process models were used and which course topics were covered. Similarly, Tenhunen et al. [14] performed a systematic literature review of software engineering capstone courses, where they categorized courses on characteristics such as team size, type of client, and duration.

6   Final Reflections and Future Work

Several important points came out during the capstone. These are described below as communication, team support, and infrastructure.

6.1 Communication

The most important challenge in Spring 2024 was with communication, including between teammates on a single team, between student teams, and with the client. Anecdotally, communication on individual teams went well, especially since students had time each week to work together in class, but there were times during the semester when teams struggled with communication specifically about tasks to do outside of class. This seemed to get better during the semester as students become more familiar with one another and with the task tracking tools on GitHub. Providing online training on how to use these tools, and introducing them in more detail in class, may help the next teams collaborate more effectively from the start. For communication between teams, the students that responded to the survey though there could be more communication than there was (2.43, with 3.0 being ideal). This coming semester, class time for these inter-team meetings will be explicitly added to the schedule.

The main communication struggles were with the community partner. Expectations with the client need to be more clearly set up front to ensure they are available at the right times, and students need more guidance about when to use the different communication channels they have available with the client. Communication challenges, especially limited face to face communication opportunities, were the main pain point mentioned by the students. An important goal for Fall 2024 will be to find the right balance for both the students and the clients. We plan to coordinate with the clients before the start of the semester to ensure regular meeting dates are already on the calendar before students again start to work on the projects.

A related point is that non-technical clients may need additional scaffolding around how to give effective feedback to student teams during these meetings. It can be challenging for clients that have never been involved in a software project to precisely formulate their needs, comment on completed work, or select from different alternative feature options. The students found it effective to offer options from a limited collection of alternatives, since this helped the clients to know what was possible and what tradeoffs were available. It may make sense to create more guidance for the clients in the future, and (an idea from our meeting with an instructor at another university) to separate out presentations into technical presentations (for the instructors and the other students) and non-technical presentations (focused specifically on the clients).

6.2 Team Support

Although all the teams worked well together this past semester, we could imagine conflicts arising on future teams. To help with this, we plan to use a tool like CATME [11, 15] to allow students to give one another peer feedback dur-

ing the semester and to raise concerns with the instructor. Student teams may also need more hands-on support in using Scrum, breaking down work into reasonable pieces, and focusing on high-priority issues for the clients, although this went reasonably well in Spring 2024.

6.3 Infrastructure

Finally, more focus needs to be placed on infrastructure requirements for student projects. Some of this is non-technical infrastructure: although triggered by a client approaching us, to keep this section active we will most likely also need to recruit and manage clients from the community. Looking at technical infrastructure, to fully deploy the software the student teams created for our community partner, we would need servers to host the website and API server, and would also need accounts with Google and Apple to publish apps to the app stores. While university-provided versions of these resources work during development, at some point client-owned versions of these resources will be needed. If students are working on multiple teams with dependencies between the teams (e.g., an API server with related clients, each developed by different teams), determining where to host projects needed by other teams needs to be done early in development to prevent problems later on, where student projects have inter-project dependencies that block progress.

6.4 The Future

In this paper, we presented our initial experience with adding a civic engagement section of our current capstone course. The student teams were able to make good progress on their projects viewed as individual projects, but the work of joining these projects into a single portal will mainly fall to the Fall 2024 cohort. We've identified communication as the most important issue to focus on going forward, but also pointed out other potential challenges above.

All artifacts created by the student teams, such as weekly status reports, the final project reports, and all presentations, are available to future teams on a shared Google Drive, while all code and project boards are hosted on GitHub. Fall 2024 will be the test of whether this is sufficient for onboarding new student teams that will have the (to students, often quite novel) challenge of reading, understanding, and modifying code that they did not create themselves. We look forward to seeing what challenges—hopefully foreseen, potentially unforeseen—arise.

References

[1] P. Armstrong. *Bloom's Taxonomy*. 2010. URL: https://cft.vanderbilt.edu/guides-sub-pages/blooms-taxonomy/.

[2] Grant Braught et al. "A Multi-Institutional Perspective on H/FOSS Projects in the Computing Curriculum". In: *ACM Trans. Comput. Educ.* 18.2 (July 2018). DOI: 10.1145/3145476.

[3] Edward J. Coyle, Henry G. Dietz, and Leah H. Jamieson. "Long-Term Community Service Projects in the Purdue Engineering Curriculum". In: *Proceedings of the 1996 American Society for Engineering Education Conference*. 1996.

[4] Janet Davis and Samuel A. Rebelsky. "Developing Soft and Technical Skills Through Multi-Semester, Remotely Mentored, Community-Service Projects". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE '19. Association for Computing Machinery, 2019, pp. 29–35. DOI: 10.1145/3287324.3287508.

[5] J. Dewey. *Experience and Education*. New York: Touchstone, 1938.

[6] P. Freire. *Pedagogy of the Oppressed*. New York: Continuum, 1970.

[7] Robert F. Dugan Jr. "A survey of computer science capstone course literature". In: *Computer Science Education* 21.3 (2011), pp. 201–267. DOI: 10.1080/08993408.2011.606118.

[8] D. A. Kolb. "Experiential Learning: Experience as the Source of Learning and Development". In: *Journal of Business Ethics* 1.15 (1984), pp. 45–57.

[9] Stan Kurkovsky et al. "Scaffolded Projects for the Social Good: A Strategy for Deploying Studio Model in CS Education". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2024. Association for Computing Machinery, 2024, pp. 1706–1707. DOI: 10.1145/3626253.3635487.

[10] Panagiotis K. Linos, Stephanie Herman, and Julie Lally. "A Service-Learning Program for Computer Science and Software Engineering". In: *SIGCSE Bull.* 35.3 (June 2003), pp. 30–34. DOI: 10.1145/961290.961523.

[11] Misty L. Loughry, Matthew W. Ohland, and D. DeWayne Moore. "Development of a Theory-Based Assessment of Team Member Effectiveness". In: *Educational and Psychological Measurement* 67.3 (2007), pp. 505–524. DOI: 10.1177/0013164406292085.

[12] William C. Oakes, Edward J. Coyle, and Leah H. Jamieson. "EPICS: A Model of Service-Learning in an Engineering Curriculum". In: *Proceedings of the 2000 American Society for Engineering Education Conference.* 2000.

[13] Suzanne Savanick et al. *Service Learning.* URL: https://serc.carleton.edu/introgeo/service/index.html.

[14] Saara Tenhunen et al. "A systematic literature review of capstone courses in software engineering". In: *Information and Software Technology* 159 (2023), p. 107191. DOI: 10.1016/j.infsof.2023.107191.

[15] Purdue University. *CATME: Smarter Teamwork.* https://catme.org/.

[16] Purdue University. *Welcome to EPICS.* https://engineering.purdue.edu/EPICS.

[17] Wikipedia. *Engineering Projects In Community Service.* https://en.wikipedia.org/wiki/Engineering_Projects_In_Community_Service.

Ungrading in Computer Science: A Case Study*

Jean H. French, Crystal K. Cox, and Michael A. Murphy
Department of Computing Sciences
Coastal Carolina University
Conway, SC 29528
`{jennis,crystal,mmurphy2}@coastal.edu`

Abstract

The ungrading movement seeks to shift student focus from earning grades to meaningful learning by replacing numerical and letter grades with actionable feedback. In this study, three upper-level computer science courses in system architecture, web application development, and software engineering were modified to use ungrading strategies by replacing weighted numerical grading with detailed feedback to encourage continuous improvement. The effectiveness of the selected methods showed mixed results across the three courses. Asking students to focus on the learning process instead of fixating on grades was found to run counter to the ingrained educational mindset with which they are familiar. Additional research will be needed to develop techniques to support learning for intellectual discovery and to build students' self-regulation skills.

1 Introduction

An educational movement called *ungrading* seeks to realign students to focus on learning for its own sake instead of simply churning through coursework in order to earn a grade. Ungrading can encompass many different strategies, but the basic idea is that instead of assigning a grade to each assignment, the instructor provides actionable, judgment-free feedback without a grade, rating, or ranking. The student is encouraged to incorporate the feedback and resubmit an improved assignment. The main characteristic of ungrading is that

---

nothing is graded while the student is engaged in the process of learning. Traditional numeric grading on activities throughout the semester has been shown to contribute to inaccurate, biased grades that do not motivate or contribute positively to the learning process. [4]

For this study, three colleagues at a public comprehensive university of approximately 10,000 students redesigned one of their courses to incorporate a set of ungrading strategies. Each removed the traditional grading scheme based on weights and categories for all activities and replaced it with a simpler, ungrading alternative to measure differences in student success, semester flow, and faculty workload. While some proponents of ungrading aim to eliminate all ranking, rating, judgments, and final evaluation from instructors, the scope of this study was a bit more modest in that it eliminated all numeric grades during the semester but still assigned a final letter grade at the end of the semester in accordance with university policy. The objective of each approach was to have students focus on the learning process itself, incorporating instructor feedback for continuous improvement on their work throughout the semester, until competency was achieved.

The remainder of this paper is organized as follows: related work is discussed in Section 2. Each of the three case studies, including results from each study, follow in Sections 3, 4, and 5. These case studies describe the application of ungrading to system architecture, web application development, and software engineering classes, respectively. Following the case studies, conclusions and intentions for future work are presented in Section 6.

2 Related Work

Ungrading encompasses many approaches that seek to reduce or eliminate the use of numeric grades, such as contract grading [3] and specifications grading [6], as well as student self-assessment and peer review. Because traditional numeric grading schemes are often arbitrary, biased, inaccurate, and de-motivating, ungrading strategies aim to prioritize judgement-free feedback over ratings and rankings. If the goal of grades is used to provide feedback, grades alone fail to communicate useful information, but this can be addressed through narrative feedback. Additionally, a focus on grades can result in negative actions by students (e.g. cheating, signing others in for attendance) and faculty (e.g. grade inflation which often incentivizes positive student evaluations in 'GPA booster' courses) rather than a focus on learning. [2]

A number of educators have found success with ungrading policies in prior work. Binary grading (pass/fail) has been used to reduce instructor workload and improve student attitudes toward learning in a number of upper-level computer science courses [1]. Ungrading policies have improved students' sense

of well-being in a project-based, non-major programming course, using self-grading (students determine their own grade) [8]. The self-directed projects also contributed to improved student motivation. Allowing resubmissions, providing only written feedback (without numeric grades), and allowing students to have input into their final grades resulted in significant improvements in students' feelings of intrinsic goal orientation, self-efficacy, and control of learning [9]. Significant improvements in self-efficacy and learning performance, control of learning, and task value have been observed with ungrading, but only small improvements in intrinsic goal orientation were found [5]. Much research has shown that both performance and quality of experience are positively impacted when intrinsic — as opposed to extrinsic — motivation is driving the activity. Self-determination theory suggests that the way to maintain intrinsic motivation is to support student feelings of competence, autonomy, and relatedness, all states which ungrading seeks to encourage. [7]

3 Case Study A – System Architecture

In the fall semester of 2022, an ungrading approach was tested in a junior-level system architecture course. Historically, this course employed a mixture of quizzes and assignments with standard numerical grading with final grades assigned by applying category weights to the different types of graded activities. As shown in Figure 1, this course had high student success rates in the previous four offerings, with 100% pass rates in Fall 2020 and Spring 2022, and a 95% pass rate in Fall 2021.

Three ungrading principles were adopted in the system architecture course. First, no numerical or letter grades were assigned to individual graded items in the course. Students were instead given detailed feedback about their submissions and were invited to resubmit to correct any deficiencies. Second, all parts of the course were effectively made optional by allowing the students to choose from a selection of topics. Finally, students were required to reflect on their own performance on the course activities and propose a final semester grade for themselves. To implement these principles, students were given a selection of five possible assignments covering logical circuits, discrete mathematics, central processing units, operating systems, and data centers. Students had to complete at least four of the assignments, which required submitting a recorded video presentation and producing artifacts to be displayed in a public portfolio.

Compared to prior semesters, student success in terms of final grades actually *decreased* during the ungrading trial, as depicted in Figure 1. In prior semesters, student success in this course (defined as passing with a grade of C or higher) averaged 95–100%. However, during the ungrading trial, this

success rate dropped to 79% even after the application of a full letter grade curve. Without the curve, the success rate would have been 43%. The fraction of students earning excellent (A) grades also decreased significantly from prior semesters.
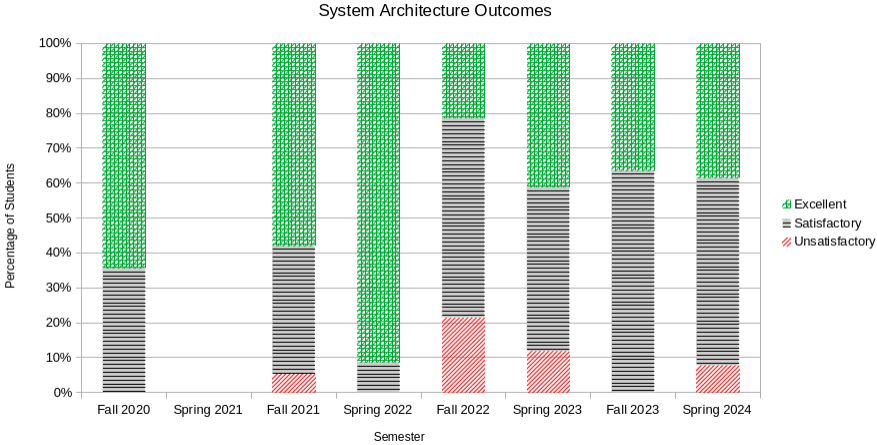
Figure 1: Grade distributions for the system architecture course, which was first offered in Fall 2020 and was not offered in Spring 2021. Ungrading was used in Fall 2022, after which the course reverted to traditional grading from Spring 2023 through Spring 2024, with plans to try another ungrading approach in Fall 2024.

To spread out the professor's workload, students were permitted to submit at most one assignment per week using a weekly dropbox in the course learning management system. Each week, the professor evaluated student submissions against the published expectations and provided detailed written feedback on both the video and portfolio artifacts for any submissions received during the prior week. Over the course of the semester, the professor observed that the students were largely ignoring the feedback, as few students took advantage of the opportunity to resubmit their work when deficiencies were observed. Students also largely ignored the instructor-provided lesson pages and recorded lectures that were linked from each assignment option, choosing instead to search the Internet on their own. Meanwhile, the professor's teaching workload for this course was significantly higher with ungrading than without, leading to a return to traditional grading during the following semester (Spring 2023). With the resumption of quizzes to motivate the students to engage with the lesson material, student success increased, albeit not quite to the levels observed

during the latter part of the COVID-19 pandemic.

4   Case Study B – Web Application Development

Ungrading was initiated in an upper-level web programming course starting in the fall semester of 2022. The course requires a background in networking, programming, and relational databases and focuses on server-side web application development. Though the individual deliverables of the course vary from semester-to-semester, a consistent requirement is the development of a dynamic web application (more specifically, a Content Management System) that covers the semester's topics. The CMS is comprehensive and can alone be used to assess programming skills learned during the semester.

Prior to ungrading, the assessments were assigned a numerical grade along with descriptive feedback. Since the programming assignments built upon prior submissions, the first task for each version was to make corrections to the prior code so that coding errors would not propagate through each new phase of development. When both feedback and grades were provided, students often ignored the written feedback. Although feedback included instructions for making corrections, students ignored the guidance and incorporated uncorrected errors into increasingly complex programs — including the final project. This inattention resulted in consistently reduced grades as each new programming assignment was evaluated.

There were two goals for implementing ungrading in this course. The first was to minimize "punishing" students with poor grades during the learning phase of the course. By removing an actual grade and only giving feedback, the intent was to motivate students to focus on continuous improvement. As the goal is for students to learn by the end of the course, the grading methods acted counter to the intent. Poor grade marks were punitive rather than encouraging students to make improvements since "grade damage" was permanent — even in the case where students mastered the material by the end of the semester. Grading in such a way was more damaging for those students taking longer to learn the material than others.

Another goal was to authentically evaluate mastery of programming skills in the class. Traditional versions of the course included weighted grades for various types of deliverables allowing multiple paths to completion. Student choice was highly encouraged in university professional development training programs. By following a multi-path to success version of the course, the significance of the grade for the final project was diluted even though the final project was the most authentic assessment of a student's programming skills. Students choosing low-hanging fruit accumulated points regardless of mastery of material. Inadequate programming skills —- regardless of a passing grade

— further sets students up for challenges and failures in more advanced classes.

In the ungrading version of the course, the student learning outcomes remained unchanged and many of the lessons and deliverables did not change significantly. Deliverables were marked either Complete or Not Complete and any non-binary assessment received detailed feedback. There were two different implementations of ungrading. In one version, students were not able to access the main portion of the final CMS project until all prerequisite assessments were marked complete. In the second version, students could access all material until the last day of the semester. In both cases, students were instructed to inform the instructor as they progressed for a review.

The final grade, required by the university, was based on the number of milestones completed in the CMS project. Milestone 1 had to be complete and correct for a grade of a C. Partially completed work would result in lower grades (D+ and lower). Additional incremental functionality increased the grade up to an A.
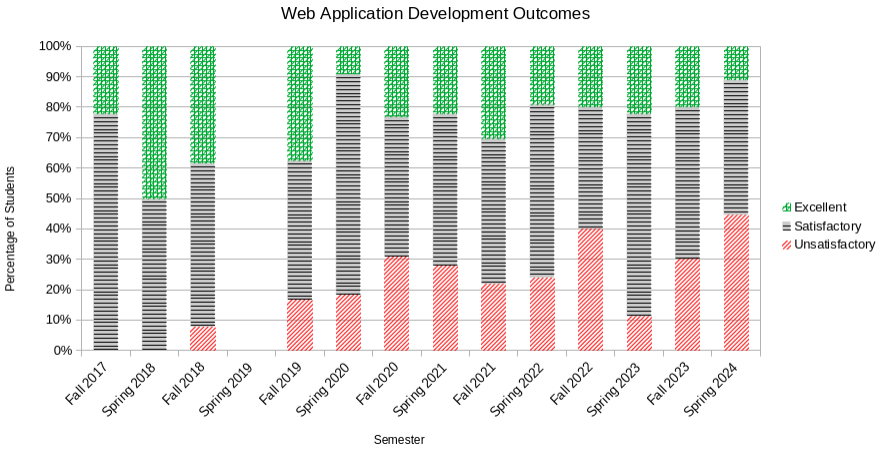
Figure 2: Grade distributions over time for the web application development course. Ungrading was used from 2022 to present.

In terms of faculty effort, there was no perceived additional work to provide only feedback since this was a regular practice. Students in the group where there were limitations on accessing the final project without completing prerequisites had a more realistic understanding of their progress. There was an opportunity to withdraw from the course if they did not make progress towards the project. One student course evaluation stated, "Not completing practice assignments should not hinder you from completing the project." In the very

next semester, the project was not restricted, yet this was problematic. The final prerequisite task was due six weeks prior to the final deadline for the CMS. There were students still working on that final prerequisite the day the final programming project was due. Students, just as in semesters prior to ungrading, attempted to piece together code at the last minute without success. This was realized in the final grades. Passing grades (C or above) dropped from 85% to 65% (Figure 2).

5   Case Study C – Software Engineering

An ungrading approach was implemented in a software engineering course for eight semesters, beginning fall 2020. Taught by the same professor as a hybrid class both before and during the ungrading trial, this course had traditionally used standard numerical grading on a 100-point scale for each activity, including coding assignments, quizzes, and presentations, with final weighted averages combined and translated into a letter grade, using a traditional scale, where 90–100 is an A, 87–89 is B+, 80–86 is B, and so on.

For the purpose of this ungrading trial, the course was redesigned into nine required modules and four optional challenge modules, each centered around a program and a written reflection as the main deliverable. Three ungrading principles were implemented. First, no numeric or letter grades were assigned to any activities during the semester, until the final grade. Second, due dates were as flexible as possible, up to the point where there would not be enough time in the semester to complete the required work, which was practically about three weeks after the original due date. Third, students demonstrated competency in this class by completing the coding assignment and writing a guided self-reflection for each assignment, with no partial credit options — all required work had to be completed to minimum requirements in order to pass the course with a grade of C. The goal of this policy was to help students develop the habits and endurance required to solve hard problems without giving up at a partial solution. Finally, while many ungrading approaches espouse detailed, actionable feedback on submitted assignments, with encouragement to students to incorporate the feedback in resubmissions, in this class feedback was mostly given only before submission, not after. Students were instructed to not submit any incomplete work, and instead ask questions and seek help as needed in order to complete the assignments satisfactorily before submitting. Using a flipped classroom approach allowed plenty of time in class for students to work on assignments and ask their questions.

Since implementing the ungrading strategies, the percentage of passing grades went up by 8% (Figure 3). Interestingly though, the percentage of A grades declined by 16% and the percentage of C grades went up by 23%.
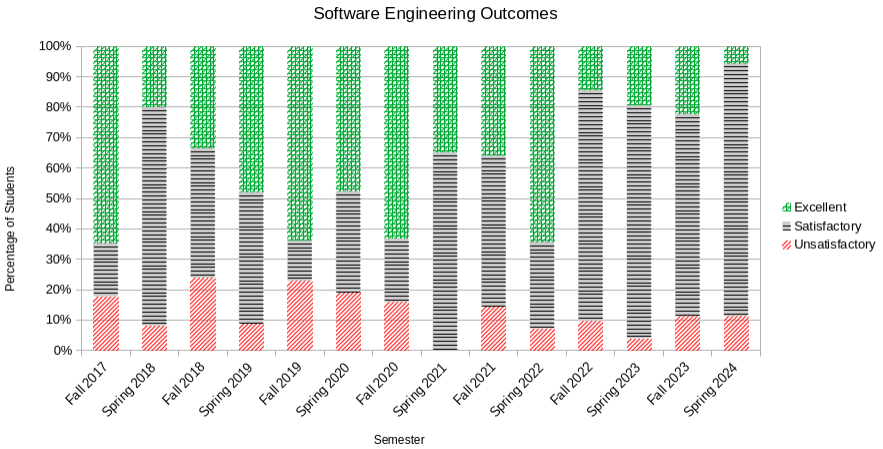
Figure 3: Grade distributions over time for the software engineering course. Ungrading was used from Fall 2020 to present.

Based on instructor observations and discussions with students, the authors hypothesize that the drop in A grades and increase in C grades is due to a couple of reasons. Firstly, the grades during ungrading are lower but more accurately reflect student competency, as there was no extra credit, late penalties, weights, or averaging that could skew grades in one direction or another. Secondly, while students appreciated the flexibility in due dates and optional assignments, it often caused them to spend their time working on assignments for other classes with hard deadlines and grades, and fall too far behind in this class to complete challenge assignments. The drop in D–F grades was likely due to both the nature of how ungrading stops penalizing the students for early mistakes, and the control that it gave students over their grade.

Aside from analyzing final grade outcomes, several observations were made regarding the impact of the implemented ungrading strategies. Most students indicated that ungrading did help to reduce their stress and anxiety, and helped them feel more in control of their grade, but that it was hard to stay on track without the pressure of hard deadlines. Students did seem to lose engagement and focus as the semester went on, and most students, regardless of grade, routinely fell behind on recommended due dates. However, because there was no partial credit for incorrect or incomplete work, most students kept working until they finished the assignments satisfactorily. As for the instructor experience, the no incomplete submissions allowed and early feedback strategy was a positive, allowing the instructor to focus on spending time partnering with

students, helping them complete the work, rather than spending that time calculating points or writing detailed feedback after submission.

6 Conclusion

The effectiveness of the selected ungrading strategies, as implemented in this study, was limited. Some results indicated worse final grades, while others showed mixed results. It is difficult to draw conclusions based solely on the grades, since the "before" grades were quite possibly less accurate reflections of student competency. It should be noted that asking students not to focus on grades is counter to the educational structure with which they are most familiar. The current standard university structure and policies do very little to support ungrading at an infrastructure level, and in fact, work against it. Even in the ungrading courses, the students still must acknowledge that there is a final grade at the end of the course.

It remains a fact that final grades exist and matter in the current university and societal environments. There are practical challenges to successfully implementing ungrading at the course level. Ungrading requires self-paced learning and this comes with its own set of challenges. A lack of engagement with the material or the inability of students to keep on schedule without the pressure of hard deadlines and grades was an issue observed in all three case studies. It will take a more concerted effort to support learning for intellectual discovery and build students' self-regulation skills. In addition to student motivation and self-regulation issues, feedback-based evaluation can have a significant impact on how much time the instructor spends on ungrading.

While the results of this initial ungrading study were mixed, it is important not to conclude that ungrading is inappropriate for computer science courses based solely on these early observations. First, the sample sizes in these courses were relatively small. Second, the COVID-19 pandemic likely inflated prior success rates due to grading considerations that were given during the international emergency. Additional data collection will be needed to determine if an ungrading approach can be successfully utilized in these courses with post-COVID students.

Future work will be needed to refine the application of ungrading to computing sciences courses and identify the techniques that are both effective and practical within the constraints of university policy. The authors plan to complete a more comprehensive study focusing on authenticity of learning, motivation, and workload (both for faculty and students), with more sensitive instruments — such as surveys and cohort data — to evaluate outcomes and address potential threats to validity. Differences in outcomes will be evaluated over multiple course modalities, including in-person, hybrid, and asynchronous

remote. Finally, techniques for adapting ungraded courses to the realities of grade-oriented transcripts and other student records will need to be explored.

References

[1] Andrew Berns. "Scored out of 10: Experiences with Binary Grading Across the Curriculum". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 1152–1157.

[2] Susan Debra Blum. *Ungrading: Why Rating Students Undermines Learning (and What to Do Instead)*. West Virginia University Press, 2020.

[3] Elmer G. Dickson. "Contract Grading". In: *Journal of Financial Education* 3 (1974), pp. 21–24.

[4] Joe Feldman. *Grading for Equity: What It Is, Why It Matters, and How It Can Transform Schools and Classrooms*. Corwin Press, Aug. 2023.

[5] Ryan Stephen Mattfeld. "Improving Student Motivation through an Alternative Grading System". In: *Journal of Computing Sciences in Colleges* 39.5 (Nov. 2023), pp. 86–95.

[6] Linda B. Nilson. *Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time*. New York: Routledge, July 2023.

[7] Richard M. Ryan and Edward L. Deci. "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being". In: *American Psychologist* 55.1 (2000), pp. 68–78.

[8] Gillian Smith. "Pairing Ungrading with Project-Based Learning in CS1 for Inherently Flexible Course Design". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education*. Vol. 1. Portland OR USA: Association for Computing Machinery, Mar. 2024, pp. 1265–1271.

[9] Scott Spurlock. "Improving Student Motivation by Ungrading". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*. Vol. 1. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, Mar. 2023, pp. 631–637.

Mind the Gap: Exploring Differences in Student Perceptions of Belonging and Inclusion in Computer Science*

Elizabeth von Briesen, Shannon Duvall,
Ryan Mattfeld, and Scott Spurlock[1]
[1]Department of Computer Science
Elon University
Elon, NC 27244
`{evonbriesen, sduvall2, rmattfeld, sspurlock}@elon.edu`

Abstract

To aid in understanding the current climate of diversity and inclusion in the Computer Science Department at our university, we developed a survey to identify and evaluate factors related to these topics. The survey was administered before the third week of participating Spring 2024 courses, which ranged from introductory classes open to any student from any major, to advanced classes taken as part of the computer science major course sequence. Our findings show that students expressed feeling more confident in their potential for success in class versus as a future professional in the field. They reported more positive interactions with faculty than with peers. Students also indicated a desire for more structured opportunities to connect with peers, and commented positively on current departmental efforts to make our common areas more welcoming. The survey results provide a baseline that we can use in the future to evaluate the impact of curricular interventions on student perceptions of inclusion and belonging. We also plan to evaluate how enhanced support for student well-being in the department and within student groups impacts perceptions and retention of underrepresented groups.

1   Introduction and Related Work

The underrepresentation of female, Black, and other marginalized identities in the field of Computer Science (CS) is a well-known, longstanding problem [4, 7]. As an example, the latest data from the National Center for Education Statistics show that women represent only 23% of majors classified as "Computer and information sciences and support services" [12, 11]. Compounding these issues, computing graduates from underrepresented groups will enter a workforce with continuing diversity gaps. In their 2021 report, Pew Research found that while women hold 50% of STEM jobs, they represent only 25% of the computing workforce and are paid less than their male counterparts across all demographics. Black and Hispanic STEM workers also remain underrepresented in the field and earn the least of all racial and ethnic groups [4].

Undergraduate computer science education is a common trajectory for individuals who wish to become professionals in the field; however, the statistics above show that this remains a leaky pipeline for students from non-majority identity groups. This problem and potential solutions are under active examination in computer science and other STEM disciplines [10, 2, 8]. Belonging and inclusion are factors often examined with respect to retention. Research has shown that a sense of belonging in introductory computer science courses is correlated with continuation of studies in the field, particularly for "minoritized students" [10]. Specific pedagogical strategies like Culturally Responsive Teaching have been shown to result in positive learning outcomes and feelings of inclusion [1]. This pedagogy has also been studied as a tool for university STEM departments to transform their culture such that it is more supportive and inclusive of *any* interested student [6]. Peer relationships are also important. Lehman et al. examined factors related to the "persistence" of women and racially/ethnically minoritized groups in computing majors, as their retention rates are lower than those of majority groups. Based on their findings, they assert that "peer experiences" are strongly related to persistence, as when these experiences are "poor," students are more likely to leave the major [7], thus leading to lower overall retention rates within programs.

Extensive literature details various methods for surveying students to understand how their identity, other personal factors, and their current environment influence their sense of belonging and inclusion. Research and related survey instruments include those developed to understand first-year experiences and feelings of belonging [5, 9], measure how belonging is beneficial in online learning [3], and determine the validity of instruments measuring general classroom community [13]. Washington et al. developed and tested a survey instrument to measure the cultural competence of computing students, identifying this as a problem requiring longitudinal study across many institutions [15]. Preceding this work, Washington explains the importance of cultural

competence and diversity in computer science education [14], working to align goals with ABET requirements. This rich body of prior work provides the basis for our survey instrument, which is focused on the measurement of students' feelings of belonging and inclusion.

In the following sections, we describe the development and details of our survey, and present the results of administering it early in the Spring 2024 semester to students enrolled in courses across our department. We discuss key themes in survey responses and conclude with some ideas about how this instrument might be used in the future.

2   Methods

For this study, we compiled a survey instrument informed by related literature to gather information on student demographics, perceptions of inclusivity in the department and classes, and attitudes about the field of computing [5, 9, 3, 13, 15, 14]. We administered the survey at a mid-size liberal arts university in 18 sections of 10 courses across our computer science curriculum, including courses for non-majors. The survey was given in each class after the drop-add period was complete and before the third week of classes.

2.1   Survey Instrument

There are three sections to our survey instrument[1]:

- **Demographics:** The first 11 questions ask for demographic information including major, year, gender identity, race, and disability status.
- **Likert Scale Ratings:** The next 16 questions ask the respondent to what extent they agree with statements about either class climate (such as "Faculty demonstrate respect for individual difference"), personal assessment ("I feel comfortable interacting with computer science professionals"), or the field of computing ("I find the field of computer science interesting"). Respondents rated their agreement with statements as either "Strongly agree", "Somewhat agree", "Neither agree nor disagre", "Somewhat disagree", or "Strongly disagree".
- **Free-Response:** The final 6 questions ask what factors are important for inclusion and what things our department in particular is doing well or could be doing better.

---

[1]Full survey: `https://github.com/muniravb/CS_UndergradSurveyBelongingInclusion`.

2.2 Questions of Note

For the remainder of this paper, we will focus on insights derived from the following subset of questions:

Likert Scale Rating:
1. Faculty in Elon's Computer Science department demonstrate respect for individual differences.
2. I feel comfortable interacting with computer science majors.
3. I feel comfortable interacting with computer science professionals.
4. I have the potential to succeed in a computer science class.
5. I expect to make important contributions in my computer science class or classes.
6. I have what it takes to become a computer science professional.

Free-Response:
7. What, if anything, makes you feel included or excluded in your computer science or STEM courses?

3 Results and Discussion

Figure 1 details the aggregate responses for Questions 1-6. For simplicity, we have combined Strongly Agree and Somewhat Agree, and Somewhat Disagree and Strongly Disagree, to "Agree" and "Disagree" respectively. This figure shows that in general, a large majority of respondents feel that faculty in our department are respectful of differences, and that they have the potential to succeed in our classes. Fewer (but still a majority) feel comfortable interacting with the student majors within our department, or are confident they have the potential for success in the tech industry beyond college.

Table 1 details student characteristics and demographics in our responses ($n = 246$).[2] The majority of our respondents (67%) were computer science majors, and all four years of study are represented. The remainder of this section breaks down the responses to determine differences between identity groups.

3.1 Finding 1: Confidence of Future Success

As shown in Figure 1, students from all demographics combined felt more confident of their success in the computer science classroom than as a computer

---

[2]Note that while the demographic categories of "Diagnosed Disability" and "Cognitive Condition" are intended to allow students to make a distinction between cognitive and physical conditions, there may be some overlap in responses.
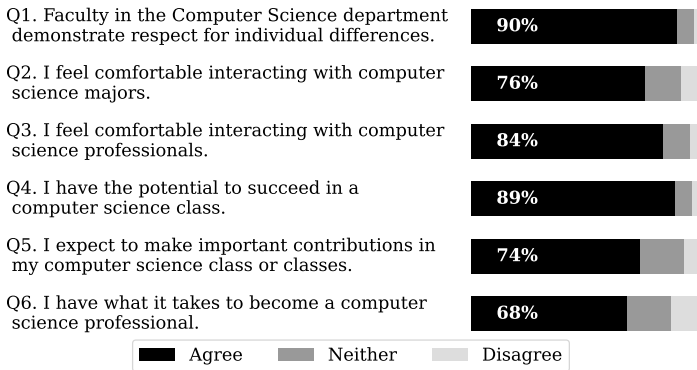
Figure 1: Student responses for selected Likert-scale questions.

Table 1: Summary of Survey Respondent Characteristics and Demographics

Factor		Number	Percent
Major	CS	149	67
	Non-CS	72	33
Year	First-Year	47	21
	Sophomore	62	28
	Junior	66	30
	Senior	46	21
Course Level	1XXX and 2XXX	115	52
	3XXX and 4XXX	106	48
Race	White/Caucasian	160	72
	African American/Black	17	8
	Hispanic/Latinx	15	7
	Asian	13	6
	Other, or prefer not to respond	16	7
Gender	Male	144	66
	Female	61	28
	Non-binary, transgender, or not listed	14	6
Diagnosed Disability	No	169	80
	Yes	42	20
Cognitive Condition	No	172	88
	Yes	23	12

science professional in the future. Overall, 89% agreed that they had potential to succeed in class, and 74% expected to make important contributions in class, but only 68% agreed that they had "what it takes" to become a computer science professional. Figure 2 shows that when these results are broken down by demographic, female-identifying students have the sharpest decline, with 90% confirming their potential for classroom success, and only 59% expressing potential for success as a professional. However, when we consider only those with a declared major in computer science, the gender difference is negligible. This means that females taking an introductory course in computer science are much less likely to believe they have the potential for success in the field at large.
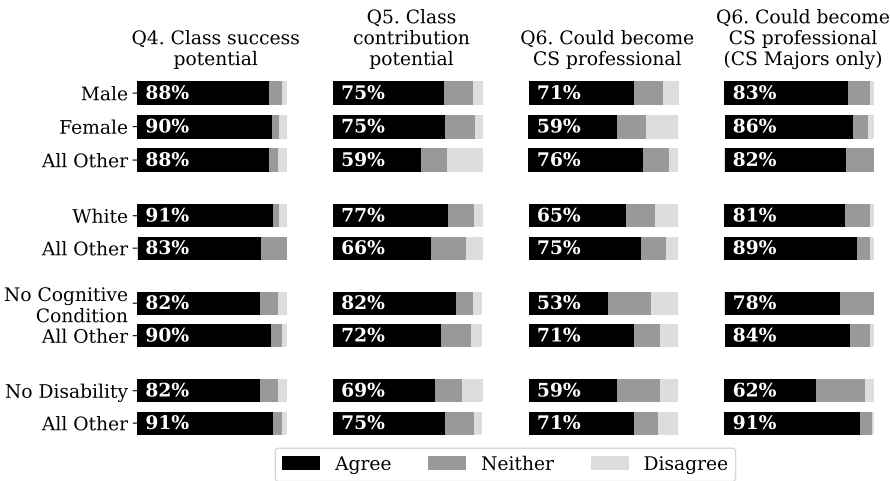
Figure 2: Student confidence of potential for success in a computer science course, expectation of making a significant impact in the course, and belief that they "have what it takes" to become a computer science professional.

Non-binary, transgender, and unlisted gendered students also showed a decline in their confidence between the classroom and the workplace, but the trend is only evident for non-majors. Compared to female students, these respondents showed a sharper decline in their confidence to make important contributions in class. While they agreed in their ability to generally do well in class (88%), many fewer agreed that they could make important contributions in the learning process (59%). This trend holds for non-white students as well. They are more confident in their potential to succeed in the class (83%) and in the workplace (75%) than in their classroom contributions (66%).

3.2 Finding 2: Peer Interactions

Students reported feeling that the computer science department faculty demonstrate respect for differences (90% agreement), and that they feel comfortable interacting with computer science professionals (84% agreement). However, they feel less comfortable interacting with computer science majors (76% agreement). As shown in Figure 3, underrepresented groups have a greater difference in their perception between faculty/professionals and their peers. For example, while 88% of individuals identifying as non-binary, transgender, or unlisted for gender feel comfortable interacting with CS professionals, only 46% feel comfortable interacting with their peers in class. Similarly, students identifying themselves as female, non-white, or disabled showed less comfort interacting with other computer science students than professionals.
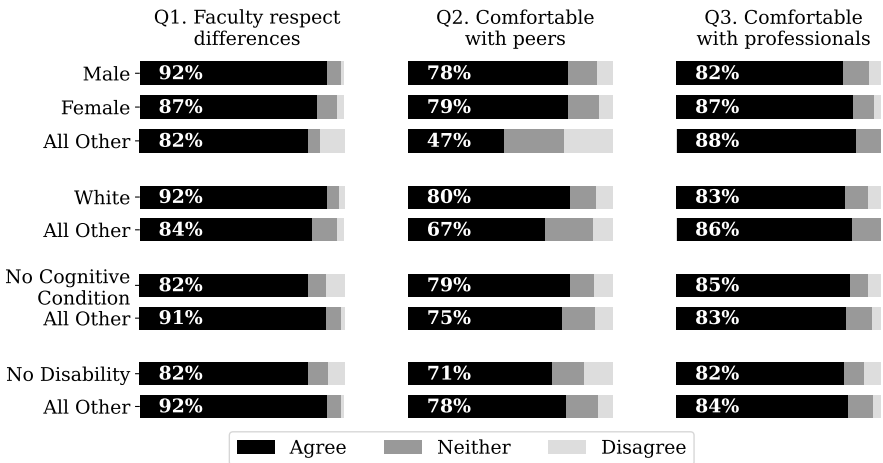
Figure 3: Student comfort with instructors, peers, and professionals.

Further evidence of this comes from the comments in the free-response questions. Question 7 asked students to describe what makes them feel included or excluded, and student responses often gave examples of either positive or negative experiences with faculty or peers in other classes. For example, one student's answer included "EVERYTIME I take a comp sci class there is some guy trying to hit on me." Another remarked that "Teachers not assuming students already know high-level stuff makes me feel included."

Figure 4 shows the results of tagging student free-response replies to this question, where they expressed either feelings of inclusion or exclusion with regards to peers, instructors, STEM or the general field of computer science,
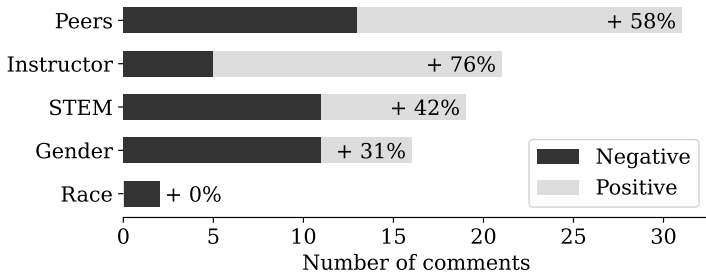
Figure 4: Number of tags across student responses to Question 7 relating to factors affecting feelings of inclusion. The number of negative comments is shown in dark gray, while positive responses are in light gray with the percentage of positive comments indicated.

gender, and race. Of responses about faculty, 76% were positive ("makes me feel included") while only 58% of responses about peers were positive. In fact, peers were paired more often with negative experiences than for any other topic, including race, gender, instructors, and STEM or field difficulty.

3.3 Finding 3: Craving Connection

Our survey included several free response questions asking about what things our department does that promote feelings of inclusion, and also for recommendations for changes or enhancements. We received positive feedback regarding our courses and environment, with one student noting that "[t]he Duke building is an awesome way to meet, hang out, and learn in. It's nice having a building with lots of fun activities like word problems, chess, sticker, places to draw, and more." Student responses also reflected the fact that they desire more structured opportunities to connect with their peers, both inside and outside of class. For example, students listed both clubs and events both as things that our department is doing well, and also as recommendations for doing more.

Perhaps most surprisingly, several students asked for more group work to be incorporated within courses. Traditionally, our students have not enjoyed working in groups, and when given the choice to pair-program, most decline. Other suggestions for improvement were to consistently have in-class student introductions and to facilitate creation of study groups. The latter is a type of peer connection historically initiated completely by students outside of class; however, these results indicate that those connections are no longer happening organically, and are still desired by many students.

4   Conclusions and Future Work

While the preceding discussion of findings centered on some of the surprising results from this study, we also noticed expected trends. Students have provided very positive, yet informal, feedback on our department's efforts to be more inclusive, and these were affirmed in our survey results. Students indicated that there are several things our department is doing well, which we can recommend to other departments trying to create a culture of inclusion:

- Train faculty on best practices for inclusive teaching, such as determining and using appropriate pronouns for students and using a diverse set of examples and images.
- Facilitate student organizations and events around computing.
- Create fun and inviting common areas. Our common area includes puzzles, games, food, stickers, and art, all of which facilitate community building and interaction.
- Experiment with alternative grading and classroom management strategies that give choices to the students.

We were encouraged by positive student feedback on these points; however, we are also interested in identifying areas needing improvement. When we began this work, our fundamental aim was to increase our students' feelings of inclusion in our computer science classes by creating curricular materials that are culturally relevant or showcase diversity in the field. The results of this study suggest more focus may be needed outside of our classrooms. Most students feel confident that they can do well in a classroom setting, but less so in a professional one. They have positive interactions with faculty, but often feel less accepted by their peers. The students' responses suggest that we need to spend more time thinking about what happens outside our classroom walls: in internship experiences, study groups, and social gatherings.

This finding presents a difficult and unfamiliar challenge for faculty, as interactions beyond our classrooms are further outside of our sphere of influence (and, often, comfort zone). Students are asking for more structured connections with their peers, such as with more group work assignments, while also acknowledging that these interactions are more likely to be negative. Students show the need to be able to see themselves as potential professionals, although encouraging internship or other "real world" experiences may backfire, since students may be faced with existing inequities in the industry. Further, we observed what appears to be diminished confidence and agency in establishing and strengthening peer relationships. Many students requested faculty facilitation of connecting with peers, which students previously were able to create for themselves. This does not appear to be specific to the computer science

department or people of underrepresented groups. Rather, it seems to reflect a general decline in students' interpersonal skills.

Combined, these factors show that a larger undertaking is needed beyond improved curricular approaches and materials. We need to contemplate how to teach our students the non-computing skills they need to be successful in their computing careers: to treat each other with respect, to be resilient in handling disrespectful interactions, and to confidently create relationships with one another. Attraction and retention of a diverse undergraduate computer science student population is one key to repairing the leaky pipeline to our profession. We plan to build on our successes, while simultaneously focusing on areas of weakness, in order to make our departmental climate one in which students from any background or identity can find success and belonging.

Acknowledgements

References

[1] Christy M Byrd. "Does culturally relevant teaching work? An examination from student perspectives". In: *Sage Open* 6.3 (2016), p. 2158244016660744.

[2] Susannah C Davis et al. "Engineering climate for marginalized groups: Connections to peer relations and engineering identity". In: *Journal of Engineering Education* 112.2 (2023), pp. 284–315.

[3] Daniela K DiGiacomo et al. "The benefits of belonging: Students' perceptions of their online learning experiences". In: *Distance Education* 44.1 (2023), pp. 24–39.

[4] 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). URL: https://www.pewresearch.org/social-trends/2021/04/01/stem-jobs-see-uneven-progress-in-increasing-gender-racial-and-ethnic-diversity/.

[5] Marybeth Hoffman et al. "Investigating "sense of belonging" in first-year college students". In: *Journal of College Student Retention: Research, Theory & Practice* 4.3 (2002), pp. 227–256.

[6] Angela Johnson and Samantha Elliott. "Culturally relevant pedagogy: A model to guide cultural transformation in STEM departments". In: *Journal of microbiology & biology education* 21.1 (2020), pp. 10–1128.

[7] Kathleen J. Lehman et al. "Nevertheless, They Persisted: Factors that Promote Persistence for Women and Racially/Ethnically Minoritized Students in Undergraduate Computing". In: *Computer Science Education* 33.2 (Apr. 2023), pp. 260–285. ISSN: 17445175. DOI: 10.1080/08993408.2022.2086401. URL: https://www.tandfonline.com/doi/abs/10.1080/08993408.2022.2086401.

[8] Yangqiuting Li and Chandralekha Singh. "Inclusive learning environments can improve student learning and motivational beliefs". In: *Physical Review Physics Education Research* 18.2 (2022), p. 20147.

[9] Jennifer Morrow et al. *Revised Sense of Belonging Scale Hoffman, M.B., Richmond, J.R., Morrow, J.A., & Salomone, K. (2002-2003). Investigating "sense of belonging" in First-Year college students. Journal of College Student Retention, 4(3), 227-256.* Nov. 2002.

[10] Sukanya Kannan Moudgalya et al. "Measuring students' sense of belonging in introductory CS courses". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education.* 2021, pp. 445–451.

[11] National Center for Education Statistics. *Bachelor's degrees conferred by postsecondary institutions, by field of study: Selected academic years, 1970-71 through 2021-22.* URL: https://nces.ed.gov/programs/digest/d23/tables/dt23_322.10.asp.

[12] National Center for Education Statistics. *Bachelor's degrees conferred to females by postsecondary institutions, by race/ethnicity and field of study: Academic years 2020-21 and 2021-22.* URL: https://nces.ed.gov/programs/digest/d23/tables/dt23_322.50.asp.

[13] Alfred P Rovai. "Development of an instrument to measure classroom community". In: *The Internet and higher education* 5.3 (2002), pp. 197–211.

[14] Alicia Nicki Washington. "When twice as good isn't enough: The case for cultural competence in computing". In: *Proceedings of the 51st ACM technical symposium on computer science education.* 2020, pp. 213–219.

[15] Alicia Nicki Washington et al. "On Measuring Cultural Competence: Instrument Design and Testing". In: *2023 ASEE Annual Conference & Exposition.* 2023.

Evaluating the Cognitive Level of GPT Models in Mathematics*

Jacob Evans, Cody Goldschmidt, Yilian Zhang
Computer Science, Engineering & Mathematics
University of South Carolina Aiken
Aiken, SC 29801
{jre5, codymg, yilianz}@usca.edu

Abstract

The current trend of using AI-based applications in everyday life has gained momentum among the general public. The GPT model has been promoted as a math tutoring tool for K-12 students. We highly question this promotion and believe a thorough examination of the GPT model's capability in mathematical cognition is necessary before it can be considered a reliable tutoring tool. In this paper, we present our preliminary findings on the GPT model's cognitive ability in mathematics. The model exhibits a low level of mathematical cognition and lacks training in important areas of trigonometry. The GPT model has not reached a level of reliability required for tutoring tool. Guardrails must be implemented for further use. We have developed an efficient strategy that allows the GPT model to categorize problems based on topic, and this self-feedback can be used to guide its problem-solving process.

1   Introduction

The recent development of LLM-based Generative AI systems has generated strong interest from the public. LLMs offer practical applications such as a writing assistant, coding pilot, and information retrieval for everyday use. OpenAI's ChatGPT is a widely used large language model (LLM) that has gained significant popularity. OpenAI and Khan Academy have promoted GPT-4 as a math tutoring tool [6] . It has been

said that GPT-4 has passed multiple AP exams and achieves high scores on SAT exams, even though LLM models usually lack math reasoning, skills, and understanding of complex problems in mathematics. Mathematics requires accuracy, precision, and reliability. An AI math tutor needs to present a high level of math understanding to teach well. We are wondering whether this is the case with the current GPT model. A thorough evaluation is necessary before using it as a reliable math tool.

In the past, researchers have noticed the non-deterministic nature of LLM models, and the reliability of their output is inconsistent. For example, a study on ChatGPT code generation [7] revealed a high degree of non-determinism. Similar non-deterministic patterns were frequently observed in mathematical computation by researchers. OpenAI introduced Wolfram Alpha (WA) and Code Interpreter (CI) plugins to address problems in math-related tasks. However, recent research [3] found that failure to formulate a valid question for these plugins resulted in invalid answers. Furthermore, beyond accuracy issues, it is important to assess whether the model presents a high level of mathematical cognition during the problem-solving process.

In [5], the authors state that we can consider math as pure symbolic logic, the relationships between objects in the real world, or the culturally constructed reasoning systems. In most perspectives, however, mathematical knowledge is rooted in our shared culture. Humans slowly learn math through guided practices designed by more experienced members and interactions with the natural environment. Mathematical cognition is a consequence of experiences; neural network models can gain this ability from gradual connection-strengthening processes. Much research has demonstrated GPT models' problem-solving ability in math, yet few experiments have been done measuring their cognition level. Our preliminary tests show the limitation of the GPT model's cognitive ability.

In this paper, we investigate the accuracy of the GPT models in problem-solving, particularly in trigonometry. We also identify the topics that lack training in the GPT-3.5 model. In addition, we test high cognitive tasks such as categorization and provide a realistic estimate of the math cognition level of GPT. Our future research plan is to develop efficient execution guardrails for the GPT model based on our results, thereby achieving high reliability in specific mathematical domains.

2 Methodology

2.1 Dataset Selection and Issues

We evaluate the MATH dataset [4], a benchmark dataset used by numerous studies. It contains 12,500 competition mathematics problems from national math competitions–such as AMC10, AMC12, and AIME. Compared with other datasets such as DeepMind Math and Khan Academy Math, the MATH dataset is much more challenging. The accuracy of the original GPT model in precalculus is in the range of 0–10%, and

Table 1: Cognitive Level

Level	Bloom taxonomy	Interpretation in mathematics
6	Create	Create new mathematical theorem/theory based on existing knowledge.
5	Evaluate	Justify decisions, thoroughly check the process, make judgments, and select the best information to explore.
4	Analyze	Predict the problem-solving process, identify theorems needed in the mid steps, recognize the pattern, generate the conclusion.
3	Apply	Apply more than one definition or formula appropriately to solve the problem.
2	Understand	Understand the question and use appropriate formulae or procedures without modification to solve the problem.
1	Remember	Apply the given formula and find the solution.

the improved model [9] is in the range of 20%; this makes it a good candidate for measuring cognitive ability. We focus our tests on precalculus trigonometric questions, a category most challenging for GPT models.

During testing, we noticed quite a few issues with this dataset. First, the difficulty level assigned to each problem does not align with the true complexity of the problem. The difficulty levels of the problems are determined by the level of the competition where they are originally from. Questions from high levels of competition tend to be more difficult, but this is not always true. The second issue is that the problem set lacks variations in topics. The set extensively covers trigonometry while overlooking other precalculus topics. With a limited number of problems, obtaining statistically meaningful data on other topics would be difficult, so we decided to focus solely on trigonometric problems.

2.2 Cognitive Complexity Level

Since the original level of the dataset reflects neither the conceptual nor cognitive difficulty, we redesigned and redefined the cognitive level of the problems we tested. We used the cognitive levels from Bloom's Revised Taxonomy model in [2] as our basis for cognition level measurement and adapted it for mathematics (Table 1).

Our trial experiments quickly show that determining the cognitive level of the GPT model based solely on its ability to answer questions correctly is inappropriate. We found that GPT's ability to solve math problems is linked to the specific topics and

the number of steps required to reach the solution. This led us to explore Achieve's framework on mathematical levels of complexity [1]. This framework establishes three types of complexity: Procedural, Conceptual, and Application. Observations suggest that both conceptual complexity/cognitive complexity and procedural complexity affect the accuracy of the GPT model. For example, GPT-3.5 struggled with problems that required a more lengthy procedure to solve, rather than those with a shorter solution.

3   GPT-3.5 Accuracy testing

3.1   Trial experiments

Our trial experiments were conducted through OpenAI API. The model we are using is GPT-3.5-turbo. The observations are listed in Table 2, where n represents number of questions. During each test, the human tester predicts whether the model can solve the problem correctly. If the model answered incorrectly, a hint would be given to guide the GPT model in the correct direction.

Table 2: Accuracy Results

Metric	Accuracy (%)
Overall accuracy (n=44)	43.2
Prediction accuracy (n=44)	70.5
Cognitive Level Accuracy	
Level 1 (n=12)	83.3
Level 2 (n=6)	50.0
Level 3 (n=11)	12.2
Level 4 (n=15)	26.7

Overall, the GPT model's accuracy decreases as the cognitive level increases. However, our detailed problem-by-problem analysis reveals that the topics involved have a greater impact on accuracy.

Another interesting observation in Table 2 is that our human tester can predict with high accuracy of 70.5% whether the GPT model will correctly solve the problem, while GPT models could not recognize whether a question is too difficult for them to solve (in Table 6)

Recognizing and evaluating problems are among the most important mathematical cognitive skills. The gap between human testers and GPT-3.5 raises further doubts about the cognitive capabilities of the GPT model.

3.2 GPT-3.5 Performance observations and statistic data

In trial experiments, we analyze each problem-solving process in detail and notice two interesting facts.

GPT-3.5 demonstrates high proficiency in some topics but shows noticeable weaknesses in other areas:

1. Trigonometric Identities: GPT-3.5 struggles to identify and apply the correct trigonometric identities, leading to a low success rate in problems related to this topic, while human users can recognize the correct trigonometric identities based on problem format and terms. GPT-3.5 fails to make these connections between the problems and formulas. For example, in Figure 1, GPT-3.5 failed to solve Problem 359 because it was unable to recognize the double angle identity was necessary to solve this problem.

2. Finding Common Denominators: Even though GPT-3.5 could find the correct common denominator, the models will cut corners or omit terms during the calculation process. In the end, the GPT-3.5 model fails to perform fraction-related operations.

3. Limited understanding of trigonometric function ranges: GPT models appear to know the range of trigonometric functions but cannot apply the knowledge in inequalities.

Problem 359:
Compute the number of degrees in the smallest positive angle x such that

$$8 \sin x \cos^5 x - 8 \sin^5 x \cos x = 1.$$

Figure 1: Double angle identity example

The accuracy table is shown in Table 3, where topics 1–8 are defined in the supplementary table. Three areas, 3A: Basic Trig identity, 3B: Sum and difference identity, 6: Manipulate algebraic expressions, have the lowest accuracy percentage. We can conclude that the current GPT-3.5 model needs further training in many important areas of trigonometry.

In addition, GPT-3.5 also struggles with problems that require lengthy solutions and problems that involve multiple topics. GPT-3.5 tends to forget previous equations as the solution length increases. Further investigation is ongoing to measure the effect of procedure complexity on accuracy. For questions involving multiple topics, the accuracy of the GPT-3.5 models accuracy decreases as the number of the topics increases, as indicated in Table 4, in which n is representative of the number of questions.

Table 3: Accuracy across each category.

Topic	Correct	Incorrect	Percentage
1A	21	11	65.6%
2A	2	4	33.3%
2B	9	7	56.3%
2C	6	2	75.0%
3A	10	26	27.8%
3B	4	13	23.5%
3C	6	7	46.2%
3E	4	7	36.4%
3F	3	6	33.3%
4A	5	9	35.7%
4D	3	6	33.3%
5C	4	5	44.4%
6	6	18	25.0%
8	9	8	52.9%

Table 4: Accuracy when multiple topics are involved.

Number of Topics	Accuracy (%)
1 (n=22)	63.6
2 (n=16)	25.0
3 (n=6)	16.7

3.3 Guided prompting

Due to persistent errors across various problems, we attempt to use guided prompting techniques to help the GPT model reach the correct answer. There are several common mistakes GPT-3.5 made: going off-topic, failing to identify necessary formulas or methods, struggling with complex terms, making symbolic errors, and forgetting known facts. To handle off-topic responses, GPT-3.5 was given topic-related hints to guide GPT-3.5 back in the right direction. A hint with the correct formula name was given to help GPT-3.5 proceed to the next step, and a "simplify" hint was provided in the case of complex terms and so on. However, the accuracy improvement with hints is very limited. The model does not show an understanding of the question in the way humans do.

4 Categorization testing

Beyond problem-solving, the ability to predict the problem-solving process, recognize patterns, and make judgments is important mathematical cognitive skills in levels 4 and 5. We focus our testing on categorization, where the GPT model is asked to analyze the problem and categorize it without solving it. Our test results show that current GPT models require extensive training to do this correctly.

4.1 GPT-3.5 Categorization on level of difficulty

Our initial motivation for categorizing problems based on their level of difficulty was to program a guardrail to redirect some problems to different APIs. Our test was conducted using all levels of problems in the Math dataset. GPT-3.5 was asked to rank each problem on a difficulty scale of 1–5. The results in Table 5 show that the GPT-3.5 model is not capable of assigning the correct difficulty level.

Table 5: GPT-3.5s ranking of dataset problems by difficulty.

Dataset Level	GPT's Accuracy (%)
1 (n=38)	2.7%
2 (n=48)	0%
3 (n=49)	0%
4 (n=35)	0%
5 (n=45)	0%

Due to the unsatisfactory difficulty ranking accuracy of GPT-3.5, we adopted a more direct approach. We asked GPT to judge its ability to solve a problem based solely on analyzing it without solving it. This could potentially serve as a predictive guardrail: a negative response will result in a warning message from the system. Preliminary results on level 1 questions (Table 6) suggest that GPT-3.5 still needs improvement to assess its problem-solving ability.

Table 6: GPT-3.5s predictions for solvable or unsolvable.

	GPT Predictions		Prediction Accuracy		
Level	Solvable	Not Solvable	Correct	Incorrect	Accuracy
1	13	24	17	20	45.9%

4.2 GPT-3.5 Categorization on topics

Categorization based on topics shows more promising result. We established the topic categorization systems outlined in Table 8. Again, our initial tests on topic categoriza-

Table 7: Categorization on topics

Level	Accuracy
1	84.2%
2	50%
3	55.6%
4	80%

tion did not meet expectation. After we use system prompt messages to guide the GPT model to select the correct category, the accuracy improved. The results in Table 7 demonstrate accurate topic-based categorization for some problems. This could lead to an efficient guardrail where GPT determine the category, and based on the accuracy results for that category in Table 3, new procedures could be developed to handle these challenging situations.

5   Conclusion

Our investigation into GPT model's math problem-solving ability and mathematical cognitive ability reveals significant limitation. Despite previous claims that GPT models could serve as excellent math tools, our findings indicates that these models struggles with high cognitive level tasks, problems involving with trigonometric identities and complex algebraic expression manipulation. The model also demonstrates difficulty in categorizing problem by difficulty level and determining the solvability of the problem.

While guided prompting in topic categorization shows a reasonable successful rate, core issues need to be addressed. Although this paper focuses on GPT-3.5-turbo, our trial experiments With GPT-4o revealed similar problems. Most importantly, we did not observe any significant cognitive level improvement from GPT-3.5 to GPT-4o. It seems that the new model has been trained with more problems and solutions but lack further training on recognizing and evaluating problems.

Our research suggests future GPT models need focus on training on mathematical cognition. Solving problem correctly is only one aspect of the mathematics; recognizing and evaluating problems is equally important. Our future plan is to build guardrails based on our topic categorization strategies for current model, develop a similar symbolic approach in certain categories as in [8] and ensure the model become a more reliable tool for educational applications.

6 Acknowledgement

References

[1] Achieve. *A Framework to evaluate cognitive complexity in Mathematics Assessments*. 2019. URL: https://www.achieve.org/cognitive-complexity-mathematics.

[2] P. Armstrong. *Bloom's Taxonomy*. Vanderbilt University Center for Teaching. 2010. URL: https://cft.vanderbilt.edu/guides-sub-pages/blooms-taxonomy/.

[3] Ernest Davis and Scott Aaronson. *Testing GPT-4 with Wolfram Alpha and Code Interpreter plug-ins on math and science problems*. 2023. arXiv: 2308.05713 [cs.AI].

[4] Dan Hendrycks et al. "Measuring Mathematical Problem Solving With the MATH Dataset". In: *NeurIPS* (2021).

[5] James L. McClelland et al. *A Parallel-Distributed Processing Approach to Mathematical Cognition*. Manuscript, Stanford University, February 18. 2016. URL: https://stanford.edu/~jlmcc/papers/McCEtAl16MsPDPApproach-ToMathematicalCognition.pdf.

[6] OpenAI. *Khan Academy explores the potential for GPT-4 in a limited pilot program*. 2024. URL: https://openai.com/index/khan-academy/.

[7] Shuyin Ouyang et al. *LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation*. 2023. arXiv: 2308.02828 [cs.SE].

[8] Tri-Huu Trinh, Yonghui Wu, Quoc V. Le, et al. "Solving olympiad geometry without human demonstrations". In: *Nature* 625 (2024), pp. 476–482. DOI: 10.1038/s41586-023-06747-5. URL: https://doi.org/10.1038/s41586-023-06747-5.

[9] Yiran Wu et al. *An Empirical Study on Challenging Math Problem Solving with GPT-4*. 2023. arXiv: 2306.01337 [cs.CL].

Appendices

7 Topics related to trigonometric question in Math Dataset

Table 8: Topics used for categorization.

Topic	Description
1A	Calculate trig values for special angles (30, 60, 90, 45 degrees)
1B	Definition of trig functions with right triangle.
1C	Definitions with unit circle, sign differences in different quadrants.
1D	Radian, degree measures, and conversion.
1E	Angle and rotation.
2A	Domain and range of trig functions.
2B	Inverse trig functions.
2C	Trig function graphs with properties and translations.
2D	Graph of inverse trig functions.
3A	Basic trig identities like Pythagorean, Reciprocal, and Quotient.
3B	Trig identities like sum and difference identities.
3C	Trig identities like double angle and triple angle.
3D	Trig identities like half angle.
3E	Trig identities like sum to product formulas.
3F	Trig identities like product to sum formulas.
4A	First degree trig equations.
4B	Solving trig equations by factoring.
4C	Solving trig equations using the quadratic formula.
4D	Equations with multiple trig functions.
4E	Equations with multiple trig functions and angles.
4F	Equations with multiple angles.
5A	Area of triangles.
5B	Law of sine.
5C	Law of cosine
5D	Similar triangles.
6	Manipulating algebraic expressions.
7A	Geometry, coordinate conversion.
8	Inequality analysis questions.

BlueAI: Designing Artificial Intelligence for Environment Science and Climate Change Learning Experiences for K12 Students[*]

Gulustan Dogan[1], Elif Sahin[1], Catherine Fay Wilkinson[1]
[1], Amelia K. Moody[1]
, and Yang Song[1]
[1]University of North Carolina at Wilmington
Wilmington, NC 28407
dogang@uncw.edu, es5510@uncw.edu, catfwilkinson@gmail.com,
moodya@uncw.edu, songy@uncw.edu

Abstract

The subject of teaching artificial intelligence (AI) in K–12 settings is rapidly expanding and will significantly affect computer education. While AI is currently a required part of computing curricula at universities, there are unique challenges in incorporating AI into K–12 education. The goal of BlueAI is to prepare K-12 educators to use game-based lessons to teach computational thinking, AI, and computer science skills that will interest children while incorporating important environmental and marine science subjects. We conducted assessments where we taught lessons at two different schools, and presented our findings.

1 Introduction

Although AI has become an essential component of computer curriculum in higher education, integrating AI into K–12 education presents special difficulties. The purpose of BlueAI is to train K-12 teachers in utilizing game-based

lessons to teach AI, computational thinking and computer science skills that engage students and infuse valuable environmental and marine science content into their teaching.

University of North Carolina Wilmington, the state's coastal university, has a strong marine science focus. Faculty in the Watson College of Education, and the College of Science and Engineering (Computer Science) will serve as leads on the project and develop sample lessons aided by AI. The curriculum is geared towards educating underrepresented populations in grades 3-8 in the coastal plains of North Carolina. This project is a great tool to introduce students to science, technology, engineering, and math (STEM) with fun marine science concepts supported with computer games and hands-on activities. As well as showing how AI can be used to address global challenges such as climate change, food production, poverty, and various diseases when applied in an interdisciplinary domain [3, 1, 7].

To use AI effectively in school settings, the interdisciplinary nature of AI must be taught properly. We believe our project will be an example for other disciplines to teach their concepts with AI too. Our proposed BlueAI Project uses AI and evidence-based instructional practices to (a) remove the traditional barriers (e.g., lack of resources; geographic distance; poverty; instructional supports) that separate underserved students from participating in authentic STEM learning opportunities; (b) spark student interest in pursuing a STEM career in marine, environmental, climate and computer sciences; (c) provide students and educators with tools and training to successfully use AI in the classroom to explore marine science and climate change research questions; and (d) educate and prepare diverse populations about STEM careers while at the same time developing critical workforce development skills.

2 Related Work

Despite AI's key place in contemporary computing, little research has been done on how students can use AI systems—particularly in K–12 environments [6].

A study done by Lee et al., used block based coding to create a virtual world which students can explore to complete challenges, which centered around the declining population of yellow-eyed penguins in New Zealand[5].

Similar to our study they used AI to educate students on marine life, however ours focused specifically on image recognition, supervised learning, data collection, and how AI is currently being used to help combat the effects of climate change. They assessed the impact the lesson had using a pre and post assessment, however their assessment covered AI topics, while our assessment aimed to see if there was an increase in ability to solve coding and algorithm

problems.

Another study was done by Chiu et al., by collaborating with teachers to develop a curriculum intended to train teachers to be able to teach AI concepts in their classrooms and increase students' interest and understanding on the topics as well[2]. They used pre and post assessments to assess how the students perceived AI topics before and after the lessons and found that the curriculum was successful[2].

Although previous work has been done to introduce AI in classrooms and incorporate interdisciplinary topics, to our knowledge, no studies have been done which teach AI while also educating and inspiring students to make a difference in the world. Our study is intended to show students they can create technology to contribute to the improvement of the health of our planet.

3   Dataset

We collected data from students at two schools to determine whether or not the lesson would improve their computational thinking skills and positively impacted their sentiment towards AI and STEM careers. We used 3 Google forms to collect demographic data, a computational thinking skill assessment, and reflection questions. Each subject chose a 4 digit anonymous identification number which they entered for each form. The demographic data was used to see how the lesson impacted students from underrepresented populations. The computational thinking assessment, which was used to determine whether or not the lesson improved the students' ability to think critically, included 25 simple coding, algorithm, and logic questions. It was taken twice per student; once before the lesson and once after, so that we could see how their scores changed. The reflection questions consisted of two prompts: "Do you view AI differently than you did before this lesson? Why or why not?" and "Are you more interested in STEM topics or careers than before this lesson? Why or why not?". The intention of this questionnaire was to assess whether or not the lesson had a positive impact on the students' sentiment toward STEM careers and topics. Written consent from students' parents or legal guardians was obtained before collecting any data. Students whose guardians did not consent to their data being collected still participated in all games and activities, but none of their data was recorded.

We split the data into two sets, one containing data collected from 49 students at the middle school, and the other from 10 students at the elementary school. More students participated in the study, however we did not use data from students who failed to complete all the forms, except for the reflection questions. The first dataset contained 36 students in grade 6, and 13 in grade 8. There were 4 students born in 2009, 9 students born in 2010, 14 students born in

2011, and 22 students born in 2012. There are 26 male and 23 female students. None of them had a disability. All of their primary language were English, and one students secondary language was Thai. One was a first generation student. None of them had free lunch and their ethnicity's were listed as 'Asian', 'Black', 'Multiracial', and 'White'. The count for each were respectively 1, 1, 2, 45. Birth year and ethnicity counts are shown in Figure 1.
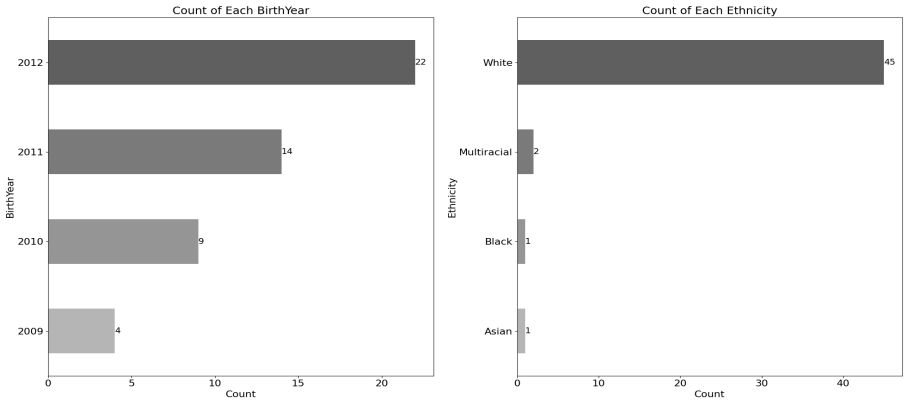
Figure 1: Dataset 1 Ethnicity and Birth Year Counts.

The second dataset is from an elementary school and contains 3 students in grade 4 and 7 in grade 5. In our second dataset there were 7 students born in 2012, 3 students born in 2013, and 2 students born in 2014. There were 6 male and 6 female students. 2 of them had a disability. All of their primary language were English, and none had a secondary language. None of the students were a first generation student. For whether or not they received free lunch, 4 of them did not and 6 were unsure. Their ethnicity's were listed as 'White/black', 'White', and 'White and Hispanic'. The count for each were respectively 1, 8, 1. Birth year and ethnicity counts are shown in Figure 2. After cleaning the data, we calculated the total scores for the computational thinking assessment using a solution key and summed the values. We compared the scores of the first and second attempts to get the difference, then merged that data with the demographics by using the anonymous id number.

4   Methodology

We developed a curriculum which contains two machine learning video games which we built using Scratch (Scratch is a project of the Scratch Foundation which is available for free at https://scratch.org.), a block-based programming

Figure 2: Dataset 2 Ethnicity and Birth Year Counts.

language, and Machine Learning for Kids, a tool made to introduce machine learning concepts which can add functioning machine learning models to coding platforms for kids. Information about sea turtle specie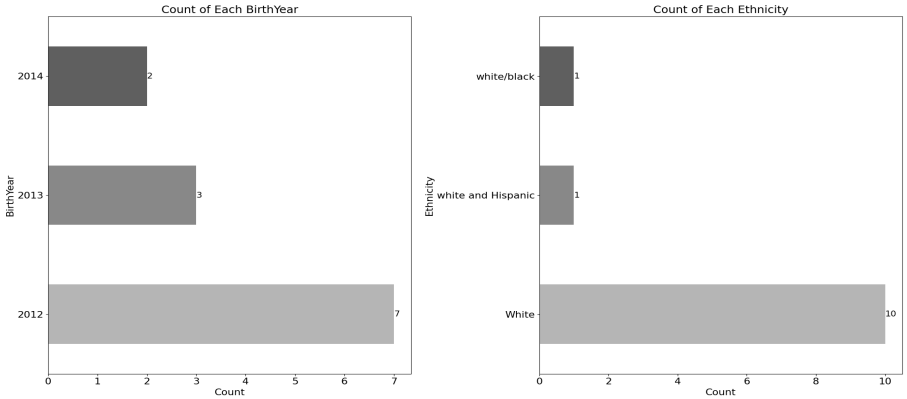s, their habitats, and how pollution impacts their populations were also covered. Instructional resources such as slide show presentations, video tutorials for setting up the game, and a lesson plan were included as well. We taught lessons using the two games at one middle school and one elementary school.

The first game focuses on sea turtle endangerment levels [4], using images of five different sea turtle species (green turtle, olive ridley turtle, hawksbill turtle, loggerhead turtle, leatherback turtle). The game involves using image recognition to sort (glide) images to corresponding endangerment levels (endangered, critically endangered, and vulnerable). We collected images of each sea turtle species from online sources and utilized Machine Learning for Kids to create a machine learning model capable of predicting if a sea turtle falls into one of the endangerment levels. The model was trained, tested, and evaluated for its predictive accuracy. Afterwards we integrated the machine learning model into code blocks in Scratch to enable the gliding of images and initiate the prediction process for the model.

The second game trains a sea turtle character to differentiate images of plastic bags and jellyfish. The game has 3 sections, "Training", "Testing", and "Pollution Info". The "Training" section is a game where the player makes the sea turtle swim to one of two images, a jellyfish or plastic bag, which is how the model is trained. While the turtle swims, there are small pollution icons floating in the water which the turtle must avoid, otherwise the player loses a life. The four icons were a plastic bag, a plastic bottle, an oil can with spilt

131

oil, and a yellow triangle with a skull and cross bones to represent hazardous chemicals.

The "Testing" button displays the turtle and one image of a jellyfish and one plastic bag, both of which are not included in the training set. The turtle automatically swims to the image it recognizes as a jellyfish and says "This is a jellyfish!". If the model is unable to recognize either image, the turtle does not move and says "I'm not sure".

Lastly, the "Pollution Info" section displayed the four pollution icons as buttons, which when clicked displayed a text box containing information about how the pollution affects the health of sea turtles and their habitats.

At the middle school, the lesson was taught to two groups of students in grade 6 and one group of students in grade 8, lasting roughly 40 minutes for each group. The teacher who allowed us to visit her classroom taught a lesson on each of the five sea turtle species a week prior to us visiting.

We began the lesson with a discussion by asking the students to define AI. After, we introduced the basic concepts of AI (such as data collection, supervised learning, training, and testing) via a slide show presentation and explained the difference between traditional computer programs and AI. We covered how AI pertains to marine science and ways it is currently being used to help with the impacts of climate change. Then demonstrated how machine learning image recognition works through the video game activities.

For the first game, students logged into pre-made accounts at Machine Learning for Kids to access a collaborative project. The project contained a section with labeled boxes where training images could be placed, each labeled as one of the three endangerment categories, endangered, critically endangered, and vulnerable. They collected 2-3 images of an assigned sea turtle species from online sources and added them to their respective categories. While they collected images, we explained the concept of supervised learning and how the machine learning model recognizes images based on the labeled training data. After the model was trained, we displayed the game on a screen at the front of the classroom. We demonstrated how to drag the machine learning code blocks, which were connected to the Machine Learning for Kids project, into the correct places in the code to implement the machine learning functionality. First we ran the game without adding any machine learning code to demonstrate that it was unable to recognize any images yet. Each image glided to the default section of the screen, the endangered category. We then asked the students to raise their hands and try to figure out where each machine learning code block should be placed to make the game function. Once the code was completed we ran the game again and showed how it was able to sort the images into the correct categories.

The second game was connected to a Machine Learning for Kids project

which had two training labels, "Food" for the jellyfish, and "Pollution" for the plastic bags. We demonstrated all three sections of the game before adding the code. When showing the "Testing" portion of the game, the turtle did not move and only said "I'm not sure.". We explained the reason was we had not collected any training data yet. Then we completed the code by having students try to determine where each block should go. One set of blocks stored and labeled the training data, another trained the model, and the last set recognized the images.

Then we had a volunteer play the game while we explained how the performance of the model depended on the data it was given. After training it once with only 2 or 3 images, we showed how it could only correctly recognize a few images when tested. Then it was trained again with more images. When the sea turtle swam to a plastic bag instead of a jellyfish, we asked students to discuss why it made the prediction incorrectly. Some conclusions were that there wasn't enough training data, the patterns in the two pictures were too similar, or the image looked very different than the images in the training set. Then we explained how each type of pollution in the game impacts sea turtle populations.

At the elementary school, we taught the lesson to one group of students in grade 3 (data is not available for this group), two groups of students in grade 4, and two groups of students in grade 5. This time we taught the lesson over the course of two days, dedicating one class period to each game. The lessons were taught the same as previously stated, except we presented a slide show covering decision trees before teaching the second game. We conducted an exercise with a decision tree used to determine the species of a sea turtle based on characteristics listed at each node. Students paired up and discussed ways to add a leatherback sea turtle to the tree by placing a question in the empty leaf node.

Following the lessons, students formed groups to discuss how AI could be used to help with the effects of climate change and were able to incorporate ideas from the lesson. One example a student came up with was to "use sensors to detect oil spills on the surface of the water". Another added the idea to "use chat bots to write an article about climate change to spread awareness".

5 Results

We calculated the average scores of the assessments before and after the lesson by each ethnicity. At the middle school, of the students who submitted both two assessment forms and a demographic form, 36 were in grade 6 and 13 were in grade 8. In grade 6, 32 students chose white as their ethnicity, 2 chose multiracial, 1 chose Asian, and 1 chose black. In grade 8, all 13 students selected
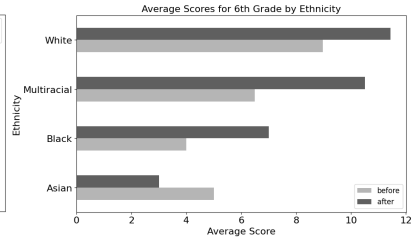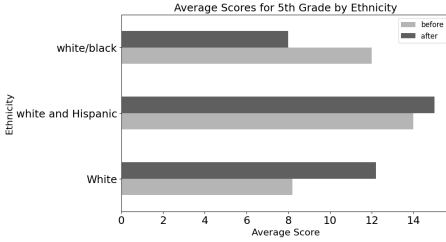
Figure 3: Average Score 5th Grade   Figure 4: Average Score 6th Grade

white as their ethnicity. Figure 4 displays the average scores by ethnicity for grade 6. For the initial test scores for this grade, white students received the highest scores on average, 8.97, and the average score increased to 11.44 for the post lesson test. However, multiracial students had the highest increase in average score, going from 6.5 to 10.5. The average score for black students went from 4.0 to 7.0, and the average scores for Asian students decreased from 5.0 to 3.0.

For grade 8, all students identified as white. The average test score had a minor increase, with the beginning average being 12.38 and the ending average as 12.62.

For grade 5, 5 students identified as white, 1 identified as white and Hispanic, and 1 identified as white/black. Displayed in Figure 3, for those who identified as white, the average score increased from 8.2 to 12.2. The average score for white and Hispanic increased from 14.0 to 15.0, and the average score for white/black decreased from 12.0 to 8.0.

For grade 4, there were 3 white students and the average score increased from 8.0 to 9.0.

We asked 2 reflection questions to the students after the lessons: "Do you view AI differently than you did before this lesson? Why or why not?" (Question 1) and "Are you more interested in STEM topics or careers than before this lesson? Why or why not?" (Question 2). The answers have been categorized to be visualized more easily and some answers fell into multiple categories. Table 1 lists the categories and their frequencies for Question 1 and Table 2 lists them for Question 2.

6   Discussion

Overall, the data indicates that the lesson resulted in an increase in computational thinking skills and positive sentiment toward AI and STEM topics. It is important to note that the middle school students completed the first assess-

Table 1: Question 1

Response Category	Count
Sees that AI is useful and can help the earth, before only saw it as a tool used to get answers from the internet	13
Was already aware of AI's capabilities, and lesson helped be more aware of it	12
Already knew about AI and doesn't have a different view	13
Already knew about AI but didn't know it could be trained with data	5
Learned more about AI's use and thinks it's creepy	2
Now knows how smart and intelligent AI is or how it can improve the world.	40
Thinks AI is complicated	5
Doesn't understand AI and is not interested in learning, thinks AI is a robot you tell to do stuff	2
Knows the coding part of AI better	15
Finds coding part of AI hard	3
Sees AI as a threat	2
Thinks AI is coding games	1

Table 2: Question 2

Response Category	Count
Doesn't think STEM for a career	41
Interested in STEM more after the class	31
Was interested in STEM before, nothing changed	11
Not sure	1
Doesn't think they understand STEM fields	1

ment on a day prior to the lesson, however the elementary school students did not get a chance to complete the initial assessment before the lesson, due to standardized testing scheduled on the days prior. They worked on it during the week the lessons were presented, then completed it a second time after the lessons were done. However, their average test scores still increased.

As for the demographic data, Asian, Black, Multiracial, White/Black, and White/Hispanic ethnicities have less samples of students that identify as such, and we believe those test scores are not from a large enough sample size to draw any significant conclusion about those groups (counts of ethnicities are shown in Figure 5 and Figure 6).

Based on the data shown in Figure 4, the lesson seemed to significantly impact the computational thinking skills for students in grade 6 the most, with Multiracial being the most positively impacted group. The average scores for 8th grade had a very small increase, which indicates the lesson did not have
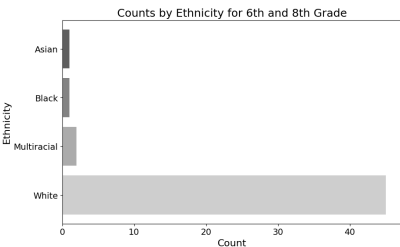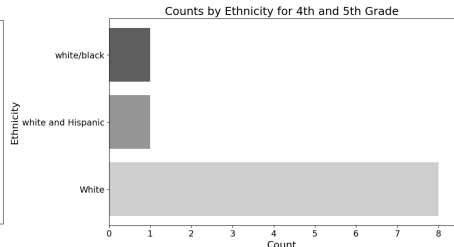

Figure 5: Grade 6 and 8

Figure 6: Grade 4 and 5

much of an impact on their computational thinking skills. Of the students in grade 5, white and Hispanic was the highest scoring group over any other grade (See Figure 3). However, the category with the greatest increase for this grade was White. Students in grade 4 scored lower than the highest scores for the other grades but still performed better after the lesson meaning it effectively improved their computational thinking skills. As for the answers to the reflection question "Do you view AI differently than you did before this lesson? Why or why not?", shown in Table 1, with "Now knows how smart and intelligent AI is or how it can improve the world." being the most frequent response category, it appears that the lesson did have a positive impact on their attitude toward AI and helped them connect the topics to real world applications. In addition, with the category "Sees that AI is useful and can help the earth, before only saw it as a tool used to get answers from the internet" appearing 13 times, it is evident that the students became aware of the positive impacts AI can have on the environment. The second most frequent category, "Knows the coding part of AI better", appeared 15 times and demonstrates that the lesson also increased the students' comprehension of the programming aspect of AI. Only 3 students gave responses which fall into the "Finds coding part of AI hard" category, and 2 students had responses categorized as "Sees AI as a threat". Based on the frequency of each category in the responses, the lesson gave a significant number of students a better understanding of how AI and coding works and how it can be used to help with the effects of climate change. Table 2 shows the frequency of each category of responses to the question "Are you more interested in STEM topics or careers than before this lesson? Why or why not?". With "Doesn't think STEM for a career" being the most frequent category, it appears that the lesson did not increase the desire to pursue a career in any STEM fields for many students, however the 31 occurrences of the category "Interested in STEM more after the class" show that there was an increase in enthusiasm toward STEM topics.

7 Future Work

In the future we aim to focus on generative AI as it is a topic which is quickly becoming integrated into our every day lives, and we believe it is important for students to understand new and emerging technology. We also hope to have teachers use our lesson to teach these topics themselves in their classrooms.

8 Acknowledgement

References

[1] Rayda Ben Ayed and Mohsen Hanana. "Artificial intelligence to improve the food and agriculture sector". In: *Journal of Food Quality* 2021.1 (2021), p. 5584754.

[2] Thomas K. F. Chiu et al. "Creation and Evaluation of a Pretertiary Artificial Intelligence (AI) Curriculum". In: *IEEE Transactions on Education* 65.1 (2022), pp. 30–39. DOI: 10.1109/TE.2021.3085878.

[3] Josh Cowls et al. "The AI gambit: leveraging artificial intelligence to combat climate change—opportunities, challenges, and recommendations". In: *Ai & Society* (2023), pp. 1–25.

[4] C Josh Donlan et al. "Using expert opinion surveys to rank threats to endangered species: a case study with sea turtles". In: *Conservation Biology* 24.6 (2010), pp. 1586–1595.

[5] Seung Lee et al. "AI-infused collaborative inquiry in upper elementary school: A game-based learning approach". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 17. 2021, pp. 15591–15599.

[6] Jiahong Su, Yuchun Zhong, and Davy Tsz Kit Ng. "A meta-review of literature on educational approaches for teaching AI at the K-12 levels in the Asia-Pacific region". In: *Computers and Education: Artificial Intelligence* 3 (2022), p. 100065.

[7] Aziza Usmanova et al. "Utilities of artificial intelligence in poverty prediction: a review". In: *Sustainability* 14.21 (2022), p. 14238.

Advisor SeaHawk: An Academic Advisor Chatbot for MSCSIS Students at UNCW*

Bulut Tok[1] and Gulustan Dogan[1]

[1]University of North Carolina Wilmington, North Carolina, U.S.A., bt6631@uncw.edu, dogang@uncw.edu

Abstract

This paper introduces Advisor SeaHawk, an advanced academic advisor chatbot for students at the University of North Carolina Wilmington (UNCW), specifically tailored for MSCSIS (Master of Science Computer and Information Science) students. Using OpenAI's GPT-4o model, Advisor SeaHawk provides personalized academic advising, including course recommendations, prerequisite checks, and detailed academic plans. The development process involves converting PDF academic records into structured JSON data, extracting student information using regular expressions, and integrating CSV-based course information. By leveraging natural language processing, Advisor SeaHawk interacts with students in a friendly manner, effectively simulating a human advisor. This chatbot aims to provide an accessible, efficient, and tailored advising experience for college students. We have not tested Advisor Seahawk yet on real student data.

1 Introduction

Transitioning into university life can make course selection and academic planning challenging, often leading to delays in traditional advising [2]. To address this, we developed Advisor SeaHawk, an AI-driven chatbot tailored for MSCSIS

students at the University of North Carolina Wilmington (UNCW), leveraging OpenAI's GPT-4o model [12].

Advisor SeaHawk provides real-time course recommendations, prerequisite verification, and academic planning. It integrates data from PDF academic records, core course information stored in a CSV file, and advanced natural language processing (NLP) techniques to offer personalized advice.

Using pandas, the chatbot reads course data (codes, names, credits, and prerequisites) from the CSV file. It also converts student academic records from PDF to structured JSON data using PyMuPDF for efficient analysis. The chatbot tracks completed, in-progress, and planned credits to ensure students meet prerequisite requirements.

Powered by the GPT-4o model [12], Advisor SeaHawk generates natural language responses to academic queries, providing personalized and accurate advice. It currently features a simple command-line interface with the potential to expand into web or mobile platforms.

In conclusion, Advisor SeaHawk revolutionizes academic advising at UNCW by making it more accessible and efficient, helping students make informed decisions about their academic paths.

2 Related Work

The Advisor SeaHawk project leverages ChatGPT to enhance academic advising at UNCW. This section reviews research on generative AI in higher education and how Advisor SeaHawk builds upon these foundations to provide tailored advising for MSCSIS students.

2.1 ChatGPT and Generative AI in Higher Education

Generative AI like ChatGPT has transformed higher education by improving learning experiences and academic advising. Akiba and Fraboni [2] found that ChatGPT provides accessible, efficient, and supportive responses, complementing human advisors and promoting educational equity. Advisor SeaHawk extends this by offering personalized advising for MSCSIS students at UNCW based on academic records.

Steele [13] highlighted both the challenges and empowering potential of ChatGPT in education, noting its capacity to enhance critical thinking and composition skills. Advisor SeaHawk utilizes these capabilities to offer tailored academic advice to MSCSIS students.

Chukhlomin [4] emphasized the need for human judgment when interpreting AI-generated insights. Advisor SeaHawk integrates AI with human expertise to provide personalized academic planning that aligns with institutional requirements.

Dai et al. [5] viewed ChatGPT as a tool that empowers students through personalized learning and improved analytics, which Advisor SeaHawk leverages for adaptive academic advising.

Banerjee et al. [3] found that custom models outperform generic LLMs like ChatGPT in providing context-specific advice. Advisor SeaHawk similarly integrates dynamic course data to deliver relevant, up-to-date advice.

Labadze et al. [9] stressed the importance of ethically integrating AI chatbots in education. Advisor SeaHawk ensures transparency, data privacy, and fairness in its academic advising.

2.2 Benefits of ChatGPT in Academic Advising

ChatGPT's ability to provide personalized learning experiences is a key benefit in academic advising. Dai et al. [5] highlighted how it can analyze educational records and offer tailored support, enhancing engagement and outcomes. Advisor SeaHawk employs these capabilities to support MSCSIS students at UNCW in their academic planning.

ChatGPT's role as an instructional assistant and virtual advisor is significant. Maniar [11] noted it can generate content, assist with writing, and provide immediate answers, reducing educators' workload. Advisor SeaHawk uses these capabilities to provide detailed, accurate advice to MSCSIS students, helping them navigate course requirements.

ChatGPT facilitates ideation and creativity by generating diverse ideas and solutions, stimulating creative thinking. However, students should critically evaluate AI-generated content. Advisor SeaHawk encourages MSCSIS students to use its recommendations alongside their own judgment and human advisor guidance.

Aithal and Aithal [1] explored ChatGPT's impact in higher education, highlighting its transformative potential. AI-driven agents provide immediate information and personalized feedback, enhancing learning experiences. Advisor SeaHawk leverages this to provide immediate, personalized advice to MSCSIS students at UNCW.

2.3 Challenges and Ethical Considerations

Integration of ChatGPT in advising raises ethical concerns. Kooli [8] examined ethical implications and solutions, emphasizing the need to address challenges to harness AI benefits while avoiding misuse. Advisor SeaHawk adheres to ethical standards, ensuring fair, transparent advising for MSCSIS students, prioritizing data privacy.

Maniar [11] discussed risks like AI-assisted cheating and over-reliance on ChatGPT, potentially reducing critical thinking skills. Institutions must up-

date policies to ensure integrity and promote responsible AI use. Advisor SeaHawk promotes responsible AI use, supporting MSCSIS students without undermining critical thinking.

ChatGPT raises issues of data privacy, fairness, and potential misuse. Developing AI literacy among students and educators is crucial. Dai et al. [5] emphasized collaborative efforts to address challenges and promote responsible AI use. Advisor SeaHawk commits to ethical standards and promotes AI literacy among MSCSIS students and educators.

Gabashvili [6] reviewed ChatGPT applications, identifying the need for interdisciplinary research to ensure responsible AI use. Advisor SeaHawk aligns with this by continually evaluating and adapting to meet ethical and practical needs of MSCSIS students.

2.4 Impact on Academic Advising

ChatGPT can significantly impact academic advising. Dai et al. [5] argued it can support advisors by complementing traditional methods with AI insights. Advisor SeaHawk leverages AI-driven insights to provide tailored advising for MSCSIS students, complementing traditional methods.

ChatGPT can facilitate ideation, helping students generate ideas and perspectives. However, students should critically evaluate AI content. Advisor SeaHawk encourages MSCSIS students to use its recommendations as a supplement to human guidance and their own judgment.

Hassani and Silva [7] discussed how ChatGPT could revolutionize advising by automating workflows, highlighting advantages and limitations. Advisor SeaHawk balances automation with human guidance for MSCSIS students.

Lekan and Pardos [10] showed AI-augmented advising provides accurate, personalized guidance. Advisor SeaHawk aligns with these findings, offering accurate advice to MSCSIS students through AI-augmented advising.

OpenAI's documentation [12] offers insights into ChatGPT's capabilities, instrumental in developing AI-driven advising tools. Advisor SeaHawk utilizes these insights to offer advanced advising for MSCSIS students at UNCW.

3 Data

Advisor SeaHawk's advising is based on comprehensive course data from a CSV file derived from the UNCW 2024 course catalog. This file includes course codes, names, credits, and prerequisites, organized in a structured format for efficient access and integration with other datasets. The tabular organization ensures quick lookups and enhances the chatbot's functionality.

As you can see in Figure 1, the table provides a clear overview of various courses along with their respective codes, names, credits, and prerequisites.

For example, the course "Artificial Intelligence" (CSC 515) requires CSC 331 with a minimum grade of B as a prerequisite, while "Machine Learning Fundamentals" (CSC 502) has no prerequisites. This structured representation of course information allows Advisor SeaHawk to accurately recommend courses to students based on their academic history and progress.

	Course Code	Class Name	Class Credits	Prerequisites
1	CSC 500	tion to Computer Science	4	None
2	CSC 502	e Learning Fundamentals	3	None
3	CSC 511	ntals of Internet of Things	3	None
4	CSC 515	Artificial Intelligence	3	CSC 331 (min grade B)
5	CSC 516	ically Inspired Computing	3	or CSC 515 (min grade C)
6	CSC 517	ibolic Artificial Intelligence	3	None
7	CSC 520	Digital Image Processing	3	CSC 340 (min grade C)
8	CSC 521	Computer Gaming	3	id CSC 340 and CSC 370
9	CSC 522	ion of Computer Systems	3	id CSC 231 (min grade B)
10	CSC 527	Quantum Cryptography	3	None
11	CSC 532	d Analysis of Algorithms I	3	CSC 500
12	CSC 533	d Analysis of Algorithms II	3	CSC 532 (min grade C)

Figure 1: Course Information

3.1 Student Academic Records

Student academic records, typically provided in PDF format, contain detailed information about their academic history, including completed courses, in-progress courses, planned courses, grades, and GPA. These records are converted into a structured format such as JSON, allowing for efficient parsing and analysis. Key information such as the student's name, department, advisor, and GPA is extracted to personalize the chatbot's responses.

3.2 Data Processing and Integration

Advisor SeaHawk processes the extracted data to calculate the student's completed, in-progress, and planned credits. This involves categorizing courses based on their status and maintaining a running tally of the student's credits. The chatbot verifies whether students meet the prerequisites for their desired courses by cross-referencing the core course information with the student's academic records.

By integrating these diverse data sources, Advisor SeaHawk generates accurate and personalized academic advice, ensuring that students receive relevant and timely guidance throughout their academic journey at UNCW. This comprehensive data processing approach enhances the chatbot's ability to provide effective and tailored academic advising for both undergraduate and graduate students, ultimately contributing to their academic success.

4   Methodology

Advisor SeaHawk employs several key components to ensure accurate and personalized academic advising for MSCSIS students at UNCW. The chatbot collects data from two primary sources: core course information and student academic records. Core course data is obtained from a CSV file of the UNCW 2024 course catalog, including course codes, names, credits, and prerequisites. This data is read using the pandas library and stored in a structured dictionary for quick access. Student academic records are provided as PDFs containing detailed academic histories, converted into structured JSON format using the PyMuPDF library. This allows efficient parsing and extraction of key information such as the student's name, department, advisor, and GPA to personalize responses.

To calculate a student's credits, Advisor SeaHawk categorizes courses into completed, in-progress, and planned courses, including grades and credits earned or expected. Prerequisites are verified by normalizing course names for accurate matching and comparing completed courses with required prerequisites, advising on eligibility or suggesting prerequisite courses when necessary. Utilizing the OpenAI GPT-4o model [12], the chatbot constructs detailed prompts with relevant student information and generates natural language responses tailored to the student's needs. This methodology ensures Advisor SeaHawk provides accurate, personalized advice, enhancing educational outcomes and supporting academic success.

Users interact with Advisor SeaHawk by uploading a PDF of their academic records and starting the chatbot application. The workflow involves the user initiating the application and uploading the document, after which the chatbot reads relevant course data from the CSV file. When the user sends a query, the chatbot forwards it to OpenAI's API, and the processed response is delivered back to the user. This loop continues until the user ends the session, at which point the chatbot sends a farewell message. This streamlined workflow ensures efficient communication, allowing Advisor SeaHawk to provide effective academic advising throughout the interaction without the need to create additional sections.

5   User Interaction and Chatbot Workflow

Users interact with Advisor SeaHawk by uploading a PDF and starting the chatbot application. As illustrated in Figure 2, the workflow begins with the user opening the application and uploading the necessary document. The chatbot greets the user and reads relevant course data from the CSV file.

The user sends queries to the chatbot, which forwards them to OpenAI's

API for processing. OpenAI generates responses that the chatbot relays back to the user. This loop continues until the user decides to end the session, at which point the chatbot sends a farewell message.

This workflow ensures smooth and efficient communication, allowing Advisor SeaHawk to provide effective academic advising throughout the interaction.

Figure 2: User Interaction and Chatbot Workflow
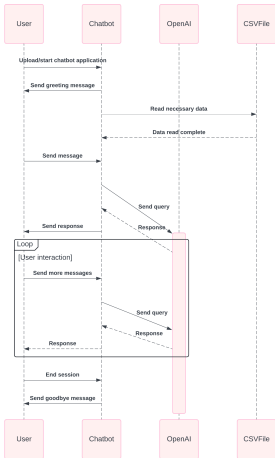
6   Results

This section outlines courses and recommendations provided by Advisor Sea-Hawk based on a student's academic history, prerequisites, and credit requirements. For a CSC student with a 3.426 GPA, the chatbot recommended starting with foundational courses without prerequisites, offering tailored academic advice based on individual records. The chatbot suggested:

```
What can I help you with today? (or type 'exit' to quit):  How many classess did I take so for bu
t not include in progress?

Response from Sammy SeaHawk:
According to the records provided, you have successfully completed the following courses:

1. **CSC 532** - Design and Analysis of Algorithms I
2. **MIS 534** - Information Security Management
3. **CSC 544** - Network Programming
4. **CSC 550** - Software Engineering
5. **MIS 555** - Database Management Systems
6. **MIS 503** - Analysis, Modeling and Design
7. **CSC 592** - Topics in Computing
8. **MIS 505** - Data Visualization
9. **MIS 506** - Text and Unstructured Data ANA

You are currently enrolled in (in-progress courses):

1. **CSC 502** - Machine Learning Fundamentals
2. **CSC 594** - Research Project (Fall 2023 and Spring 2024)

The transcript shows that the **CSC 500** course was not counted towards your degree requirements
because the limit for certain types of credits was exceeded. As for CSC 594, it is listed as both
completed ("I" grades) and still required under the heading "RESEARCH PROJECT REQUIREMENT", indic
ating it is in-progress.

So, you have completed 9 classes previously, and currently, 2 courses are in progress, indicated
by enrollments in different semesters that overlap. The CSC 500 course, despite being completed,
is not included due to credit limitations. Thus, excluding courses in progress, you have taken a
total of **9 classes** towards your degree.
```

Figure 3: Course Extraction


```
What can I help you with today? (or type 'exit' to quit):  does CSC 551 have any prerequisites an
d am I eligible to to take this course?

Response from Sammy SeaHawk:
Based on the course information provided, CSC 551: Software Assurance (3 credits) has the followi
ng prerequisites:

- CSC 550: Software Engineering

From your academic records, you have successfully completed CSC 550 with a grade of A in Spring 2
023. Therefore, you meet the prerequisite requirement for CSC 551 and are eligible to enroll in t
his course.
```

Figure 4: Prerequisites' Information

Figure 3 demonstrates the chatbot's capability to extract relevant course information from academic records. The figure shows a sample interaction where the user inquires about the prerequisites for CSC 551: Software Assurance. The chatbot effectively retrieves the necessary information, confirming that CSC 550: Software Engineering is a prerequisite and verifying the user's eligibility based on their academic records.

Figure 4 further illustrates how the chatbot handles detailed queries regarding course completion and current enrollments. The user asks about the number of classes taken so far, and the chatbot provides a comprehensive list of completed and in-progress courses. This ensures that the user has a clear understanding of their academic progress and remaining requirements.

Course Code	Course Name
CSC 500	Introduction to Computer Science (4 credits)
CSC 502	Machine Learning Fundamentals (3 credits)
CSC 511	Fundamentals of Internet of Things (3 credits)
CSC 517	Symbolic Artificial Intelligence (3 credits)
CSC 527	Quantum Cryptography (3 credits)

Table 1: Recommended Courses

Table 1 shows the courses recommended based on the student's academic history and prerequisites. For example, after completing CSC 500, advanced courses like CSC 532, CSC 544, CSC 550, and CSC 565 were suggested. Advisor

SeaHawk also highlighted courses requiring specific prerequisites, such as CSC 533 and CSC 551, ensuring students meet the requirements before enrolling.

The chatbot efficiently parses PDF academic records to extract completed credits, in-progress courses, and planned classes, accurately calculating total credits and identifying required courses. This clear guidance helps students plan their coursework and stay on track for graduation.

7 Discussion

Advisor SeaHawk showcases the potential of AI-driven academic advising by offering personalized, efficient course recommendations. Using natural language processing, it interprets student records and integrates course prerequisites and credits. Its ability to handle diverse data formats, such as PDFs, adds to its versatility. The chatbot's real-time interaction makes it useful for both undergraduate and MSCSIS students, enhancing the academic advising process with a student-centered approach.

8 Testing and Validation

The chatbot was tested with the author's academic records and sample records to ensure accuracy before it's launched for all students. This phase helped identify and address potential issues, refining the chatbot's performance.

9 Conclusion

Advisor SeaHawk delivers accurate and tailored academic advice while complementing human advisors, who are essential for handling unique cases. It enhances efficiency and accessibility in advising, supporting faculty in their roles. As the chatbot evolves with dynamic data sources, its impact on academic advising at UNCW will grow.

10 Acknowledgements

References

[1] P. S. Aithal and Shubhrajyosna Aithal. "ChatGPT for Higher Education and Professional Development: A Guide". In: *SSRN* (2024).

[2] Daisuke Akiba and Michelle C. Fraboni. "AI-Supported Academic Advising: Exploring ChatGPT's Current State and Future Potential toward Student Empowerment". In: *Education Sciences* 13.9 (2023). ISSN: 2227-7102. DOI: 10.3390/educsci13090885. URL: https://www.mdpi.com/2227-7102/13/9/885.

[3] Ronit Banerjee et al. *Customizing Large Language Models for Automated Academic Advising at Universities*. Tech. rep. Worcester Polytechnic Institute, 2024.

[4] Valeri Chukhlomin. "Exploring the Use of Custom GPTs in Higher Education Strategic Planning: A Preliminary Field Report". In: *SSRN* (2024). DOI: 10.2139/ssrn.4793697. URL: https://ssrn.com/abstract=4793697.

[5] Yun Dai, Ang Liu, and Cher Ping Lim. "Reconceptualizing ChatGPT and generative AI as a student-driven innovation in higher education". In: *Procedia CIRP* 119 (2023), pp. 84–90. ISSN: 2212-8271. DOI: 10.1016/j.procir.2023.05.002. URL: https://www.sciencedirect.com/science/article/pii/S2212827123004407.

[6] Irene S. Gabashvili. "The impact and applications of ChatGPT: a systematic review of literature reviews". In: *arXiv* 2305.18086 (2023). URL: https://doi.org/10.48550/arXiv.2305.18086.

[7] Hossein Hassani and Emmanuel Sirmal Silva. "The Role of ChatGPT in Data Science: How AI-Assisted Conversational Interfaces Are Revolutionizing the Field". In: *Big Data and Cognitive Computing* 7.2 (2023). ISSN: 2504-2289. DOI: 10.3390/bdcc7020062. URL: https://www.mdpi.com/2504-2289/7/2/62.

[8] Chokri Kooli. "Chatbots in Education and Research: A Critical Examination of Ethical Implications and Solutions". In: *Sustainability* 15.7 (2023). ISSN: 2071-1050. DOI: 10.3390/su15075614. URL: https://www.mdpi.com/2071-1050/15/7/5614.

[9] L. Labadze, M. Grigolia, and L. Machaidze. "Role of AI chatbots in education: systematic literature review". In: *Int J Educ Technol High Educ* 20 (2023), p. 56. DOI: 10.1186/s41239-023-00426-1. URL: https://doi.org/10.1186/s41239-023-00426-1.

[10] Kasra Lekan and Zachary Pardos. "AI-Augmented Advising: AI-Augmented Advising: A Comparative Study of ChatGPT-4 and Advisor-based Major Recommendations". In: *AI for Education: Bridging Innovation and Responsibility at the 38th AAAI Annual Conference on AI*. 2024. URL: `https://openreview.net/forum?id=ZHidmlDWJs`.

[11] Himanshu Maniar. "An Impact of Chat GPT in Higher Education". In: *International Journal of Innovative Science, Engineering Technology* 10.08 (2023). ISSN: 2348-7968. URL: `https://www.ijiset.com/vol10issue8/`.

[12] OpenAI. *OpenAI Documentation*. `https://platform.openai.com/docs/overview`. 2023.

[13] Jennifer L. Steele. "To GPT or not GPT? Empowering our students to learn with AI". In: *Computers and Education: Artificial Intelligence* 5 (2023), p. 100160. ISSN: 2666-920X. DOI: `https://doi.org/10.1016/j.caeai.2023.100160`. URL: `https://www.sciencedirect.com/science/article/pii/S2666920X23000395`.

Strategies for Recruitment and Retention in Computer Science[*]

Amber Wagner
Department of Computer Science
University of Alabama at Birmingham
Birmingham, Alabama 35294
awagner@uab.edu

Abstract

Once upon a time, Birmingham-Southern College (BSC), a small Liberal Arts school in Birmingham, Alabama, was looking to increase the number of underrepresented groups in Computer Science (CS), which led to participating in a National Center for Women and Information Technology (NCWIT) Learning Circle[21]. It was the Learning Circle that enabled BSC to learn more about the students and how to better support them throughout their studies. To better understand why students were taking the introductory classes and why students remained in the major through graduation, students were given entry and exit surveys. The entry survey results demonstrate that students predominantly enrolled in the introductory course out of curiosity or an interest in computing. The exit survey results show that students were most impacted by supportive faculty, interesting coursework, their own future career goals, and engaging assignments.

1   Introduction

Based on data provided by the National Center for Education Statistics[8], the percentage of students earning a Bachelor's in an Engineering-related discipline

---

is only 25% female for the 21-22 academic year, and 15% of the number of conferred STEM Bachelor's degrees were earned by Black students, and 18% of the STEM Bachelor's degree were earned by Hispanic students. While recruitment of underrepresented groups remains an issue, a more pressing issue is retention. How does a professor, department, or institution maintain a student's interest beyond the first class? As the "enrollment cliff" looms[18], retention is extremely important, not just within undergraduate programs, but to encourage students to pursue graduate degrees.

Using recommendations from an NCWIT Learning Circle, Entry and Exit Surveys were introduced to the CS program at BSC. An Exit Survey already existed; however, it did not ask questions regarding retention as recommended by NCWIT. The surveys, which were evaluated from Fall 2020 through Spring 2023 allowed the faculty to identify why students were enrolling in the introductory courses as well as what kept them motivated throughout the program. While the initial focus was on underrepresented students, the strategies applied in this study were implemented for all students due to the small numbers at BSC, and the feedback students provided demonstrate that the implemented interventions were helpful throughout the 20-21 to 22-23 academic years.

2 Related Work

Recruitment and retention in CS has been a long-standing issue. Prior to 2013, universities were eager to recruit and retain students[1, 7, 11, 14, 20] as CS enrollment declined across the US. Many of the approaches used then are still top strategies for retention such as undergraduate research[14] and peer learning[20]. The secret got out, and CS enrollment exploded; however, recruitment and retention among students from underrepresented groups is still an issue[19]. In 2018, 15 researchers as part of the Association for Computing Machinery's Education Board Retention Committee made a large number of recommendations for interventions that would positively impact retention including some presented in this paper: groupwork, CS experiences beyond the classroom, and sending students to conferences, particularly those targeting specific groups[19]. Other strategies for increased retention include focusing on career goals[16] and ensuring the classroom climate is one that allows students to make mistakes[3]. Peteranetz et al.[15] emphasizes the importance of implementing evidence-based teaching practices in the classroom as concept mastery was found to be a heavy predictor for retention, and evidence-based teaching practices can assist students in building the necessary skills to continue taking CS courses.

The survey results indicate many of the same interventions used previously to be effective regarding retention. These interventions are described in the

following section with student feedback on their perceptions of the interventions provided in Section 5.

3   Program Description

The CS program at BSC was an applied CS program (the school unfortunately closed May 2024 due to financial issues). While traditional CS courses were taught, each course attempted to be taught in an applied context either through assignments or overarching subjects. As examples, each course was project-based requiring final projects rather than final exams, computer architecture was taught through robotics, and operating systems was taught through a cybersecurity lens. Non-majors generally took either CS Principles or Programming I (taught in Python), and majors started with Programming I.

Over the three years of administering these surveys, 7.4% of students taking CS Principles chose to continue their CS studies by taking Programming I. Students were highly encouraged to pursue a minor throughout their undergraduate studies, which became the focus for many of their class projects or their senior capstone project. Students were encouraged to participate in internships as well as Research Experiences for Undergraduates through NSF and undergraduate research at BSC.

3.1   Interventions Implemented

Several interventions were implemented throughout the program during the 20-21, 21-22, and 22-23 academic years, each of which is described below.

1. The **lab space**, which was dedicated to CS students and managed by a Work Study student, was upgraded to make the space more inviting. The purpose of this lab was for students who wanted tutoring or were working on group projects or research projects to have a space where they could collaborate and have access to equipment related to their projects. The lab had a Linux computer, Windows computer, and two Apple computers, allowing students to test applications in different environments. Robotics hardware and a motion caption system were available to students. There were fun posters, comfortable laptop chairs, a large LCD for group discussions.
2. **Engaging assignments** were given throughout each course as often as possible. Assignments were either created by the faculty member, selected from online repositories (e.g., EngageCSEdu.org[9] or Nifty Assignments[2]), or a BRIDGES Assignment[4]. The BRIDGES assignments were primarily used in the data structures/algorithms and programming

language courses, allowing students to create sophisticated visualizations despite not having significant programming experience.

3. **Groupwork** was used frequently. In particular, many lessons were taught using the POGIL framework[17]. Groupwork allows students to learn from one another as well as talk about the topic being taught. Stephenson et al.[19] also recommends groupwork as an effective intervention.

4. All faculty at BSC were required to have three scheduled **office hours** per week; furthermore, faculty were highly encouraged to keep their doors open when in the office, which showed students that faculty members were present and accessible.

5. The department increased **undergraduate research** by offering independent study credit or research stipends. While open to all students, recruitment of undergraduate researchers was focused on students from underrepresented groups as part of the NCWIT Learning Circle strategic plan; however, students not from underrepresented groups were not denied the opportunity to participate. This was one of the larger initiatives because evidence shows that conducting research aids in developing one's "sense of belonging" in STEM [5] and can aid in the persistence of studying STEM fields [6, 10, 12, 13].

6. **Tutoring and Lab Manager positions** were extended to students who demonstrated skill and responsibility. They provided students in the positions with that "sense of belonging," and it gave younger students an opportunity to see what their future could hold as well as have peer mentors.

Some of the initiatives implemented had more impact than others as indicated in the survey results presented in Section 8; however, all of the initiatives were effective in some way based on course feedback.

4 Surveys

Two surveys were administered to students during the 20-21, 21-22, and 22-23 academic years. The Entry Survey was administered to all students taking the CS Principles and Programming I courses. The questions on the Entry Survey predominantly came from questions NCWIT recommended during Learning Circles meetings.

An Exit Survey was administered to graduating seniors in their final month of classes. Many of the questions were related to how well the courses met the stated Student Learning Outcomes along with a few others regarding course conflicts and elective choices, which have been removed for brevity. The relevant Entry and Exit Survey questions are listed Table 1; demographics questions have been removed for brevity.

Table 1: Entry and Exit Survey Questions

Entry Survey	Exit Survey
Which of the following experiences in programming or computing did you have before coming to BSC.	Did you participate in any out of class experiences in the wider CS community?
Why did you enroll in this course?	I feel I had adequate opportunities for out-of-class experiences in the wider CS community.
Did you receive information about CS at this institution from any of the following sources?	CS faculty members were available outside of class time.
Which of the following best describes your current or intended major?	CS faculty members were helpful in class and out of class.
If major is CS, please rank up to three factors that have most strongly influenced your decision to major in CS.	What contributed to your completion of the major? Check all that apply.
Did you enter BSC as a first-year student or as a transfer student.	Did you find the space appealing?

5   Results and Discussion

The following sections describe the results of the Entry and Exit Surveys along with their implications.

5.1   Entry Survey and Recruitment to the Major

The Entry Survey provided information regarding who took introductory courses and why, which was used to aid in recruitment efforts for the major. There were 116 responses during the 20-21, 21-22, and 22-23 academic years. Of those who responded, 53% had zero previous experience before taking one of the introductory courses, and 52% chose to enroll in a CS course out of curiosity or an interest in computing. Only 26% enrolled in the course because it was a requirement for their major. At BSC, students learned most about CS courses through their faculty advisor (29%), and others heard about the courses from other students (26%). Therefore, human connection was primarily responsible for choosing to enroll in a course.

The survey results demonstrate that many students are naturally interested in CS, and they listen to advisors and fellow students regarding course recommendations. Therefore, all advisors on campus should have information regarding introductory CS courses for their students, or CS department web-
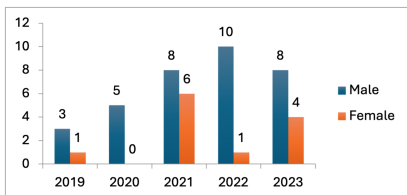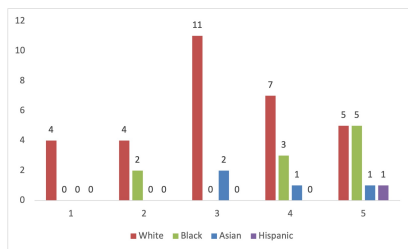
Figure 1: Grads by Gender

Figure 2: Grads by Ethnicity/Race

sites should have a space dedicated for non-majors or undecided majors to learn more about the introductory courses and why they should enroll in them.

An interesting result from the survey is students' top three factors for wanting to major in CS: personal interest in CS, high starting salary, and technology's potential for positive social impact. Faculty cannot control starting salaries, and students enrolled in our courses are already there because of their personal interest. A strategy for recruiting students from non-majors to majors might be to incorporate more assignments or news articles related to the positive social impact of CS.

5.2 Exit Survey and Strategies for Retention in the Major

BSC was a small school with a total attendance fluctuating around 1,000 students; therefore, there were few graduates each year. During the 20-21, 21-22, and 22-23 academic years, there were 28 responses to the Exit Survey. Figure 1 illustrates the number of majors who graduated each year starting in 2019 by self-identified gender, and Figure 2 illustrates the number of majors by ethnicity/race. The CS program did not begin until 2017, making the first graduating class the class of 2019.

Students participated in many out of class experiences with guest lectures and internships having the highest participation (Table 2). Stephenson et al.[19] highlights the positive effects of providing CS experiences for students outside of the classroom.

Regarding what contributed most to students' completion of the major, Table 3 shows that beyond their own career goals (just as [16] found), supportive faculty, engaging assignments, and interesting coursework had the largest impact. The student organization was not very active, which is why it likely did not have much impact on graduating seniors.

Based on the information provided in Table 3, what happens in the classroom is most beneficial towards helping students make it through the major. At

154

Table 2: Student Experiences Outside the Classroom

Out of Class Experience	Count
Attended a CS talk on campus (e.g., guest lecture)	21
Attended a CS talk off campus (e.g., attending a CS conference or colloquium at another institution)	9
Presented a CS talk or poster off campus or made a video presentation to an industry partner	7
Submitted a paper to a conference proceedings or journal or other outlet that was beyond course requirements	4
Participated in a CS contest (e.g., ACM programming contest)	2
Participated in summer research at this or another institution	6
Participated in a CS internship	16
Worked as a CS tutor	7
Served as a Teaching Assistant	3
Tutored CS off-campus	3

Table 3: Contributing factors to completion of the major.

Contributing Factor	Count
Interesting coursework	26
Engaging assignments	20
Future career goals	23
Involved student organization	2
Tutoring opportunities	5
Supportive faculty	23

the end of the survey, students were given an opportunity to provide additional feedback. While there were three comments suggesting additional coursework to offer, most of the comments regarded supportive faculty. One student stated that a faculty member in the department changed their life. Another stated that the CS department "is the best department, with the best faculty, on campus." Yet another stated, "The faculty were so caring and very helpful." These comments highlight the importance of training faculty not only what to teach, but how to teach as well as the effectiveness of an open office door.

6 Conclusion and Future Work

The Entry and Exit Surveys administered at BSC provide some insight into recruitment and retention strategies. On the recruitment side, providing information regarding why introductory courses are beneficial to non-majors/undecided majors to campus advisors as well as on a department website could be helpful

to entice more students to enroll in introductory courses. Furthermore, within those introductory courses, it's important to provide examples, through articles or assignments, of how technology can have a positive social impact. This may lead to more students choosing to pursue a CS major.

Regarding retention, supportive faculty, engaging assignments, and interesting coursework appear to have the biggest impact to students' completion of the degree. Steps to take may include having faculty participate in teacher training workshops provided by the campus or POGIL workshops[17], potentially. Providing faculty with resources such as BRIDGES[4], Nifty Assignments[2], or EngageCSEDU.org[9] among others could also benefit both faculty (by providing pre-made materials) and students (engaging assignment). Finally, ensuring faculty understand the importance of being available to students is crucial. Students' career goals was a strong motivating factor. Perhaps, discussing the types of careers students may pursue early on in the program or having guest speakers could motivate students further as suggested by [16].

In the future, repeating this survey process at a larger institution may provide different results. Areas to explore further in future students would include the impact of student organizations and undergraduate research. One undergraduate researcher stated that the experience increased their love for computing.

While other factors are important: students' career goals, participation in internships, starting salaries, and attending tutoring sessions, these are aspects we can discuss with students and encourage them, but they are not in our control as faculty. Instead, we should focus on the topics we can control: being supportive and providing interesting coursework with engaging assignments.

Acknowledgements

References

[1] Alex Aravind. "Students recruitment and retention in computer science: panel discussion". In: *Proceedings of the 16th Western Canadian Conference on Computing Education*. WCCCE '11. Prince George, British Columbia, Canada: Association for Computing Machinery, 2011, pp. 43–44. ISBN: 9781450307925. DOI: 10.1145/1989622.1989633. URL: https://doi.org/10.1145/1989622.1989633.

[2] Nifty Assignments. *Nifty Assignments*. http://nifty.stanford.edu/.

[3] Lecia J. Barker, Melissa O'Neill, and Nida Kazim. "Framing classroom climate for student learning and retention in computer science". In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA: Association for Computing Machinery, 2014, pp. 319–324. ISBN: 9781450326056. DOI: `10.1145/ 2538862.2538959`. URL: `https://doi.org/10.1145/2538862.2538959`.

[4] BRIDGES. *About BRIDGES*. `https://bridgesuncc.github.io/`.

[5] Brian A. Burt et al. "STEM Validation Among Underrepresented Students: Leveraging insights from a STEM diversity program to broaden participation". In: *Journal of Diversity in Higher Education* 16.1 (2023), pp. 53–65. DOI: `https://doi.org/10.1037/dhe0000300`.

[6] Anthony Carpi et al. "Cultivating minority scientists: Undergraduate research increases self-efficacy and career ambitions for underrepresented students in STEM". In: *Journal of Research in Science Teaching* 54.2 (2017), pp. 169–194. DOI: `https://doi.org/10.1002/tea.21341`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/tea. 21341`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/ tea.21341`.

[7] Tim DeClue et al. "Five focused strategies for increasing retention in Computer Science 1". In: *J. Comput. Sci. Coll.* 26.5 (May 2011), pp. 252–258. ISSN: 1937-4771.

[8] National Center for Education Statistics. *Undergraduate Degree Fields*. `https://nces.ed.gov/programs/coe/indicator/cta/undergrad- degree-fields`.

[9] EngageCSEdu.org. *EngageCSEdu.org*. `https://www.engage-csedu. org/`.

[10] Mica Estrada et al. "Improving underrepresented minority student persistence in STEM". In: *CBE—Life Sciences Education* 15.3 (2016), es5.

[11] Samantha L. Finkelstein et al. "SNAG: using social networking games to increase student retention in computer science". In: *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*. ITiCSE '10. Bilkent, Ankara, Turkey: Association for Computing Machinery, 2010, pp. 142–146. ISBN: 9781605588209. DOI: `10. 1145/1822090.1822131`. URL: `https://doi.org/10.1145/1822090. 1822131`.

[12] Anura U Goonewardene et al. "An interdisciplinary approach to success for underrepresented students in STEM". In: *Journal of College Science Teaching* 45.4 (2016), p. 59.

[13] Paul R Hernandez et al. "Sustaining optimal motivation: A longitudinal analysis of interventions to broaden participation of underrepresented students in STEM." In: *Journal of educational psychology* 105.1 (2013), p. 89.

[14] Joan Peckham et al. "Increasing student retention in computer science through research programs for undergraduates". In: *SIGCSE Bull.* 39.1 (Mar. 2007), pp. 124–128. ISSN: 0097-8418. DOI: 10.1145/1227504.1227354. URL: https://doi.org/10.1145/1227504.1227354.

[15] Markeya S. Peteranetz and Leen-Kiat Soh. "A Multi-level Analysis of the Relationship between Instructional Practices and Retention in Computer Science". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 37–43. ISBN: 9781450367936. DOI: 10.1145/3328778.3366812. URL: https://doi.org/10.1145/3328778.3366812.

[16] Markeya S. Peteranetz et al. "Future-Oriented Motivation and Retention in Computer Science". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE '18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 350–355. ISBN: 9781450351034. DOI: 10.1145/3159450.3159513. URL: https://doi.org/10.1145/3159450.3159513.

[17] CS-POGIL. *Process Oriented Guided Inquiry Learning in Computer Science*. https://cspogil.org/Home.

[18] David Rosowsky. *The Cliffs of Higher Ed: Who's Going Over and Why*. https://www.forbes.com/sites/davidrosowsky/2024/02/03/the-cliffs-of-higher-ed-whos-going-over-and-why/. 2024.

[19] Chris Stephenson et al. *Retention in Computer Science Undergraduate Programs in the U.S.: Data Challenges and Promising Interventions*. New York, NY, USA: Association for Computing Machinery, 2018. ISBN: 9781450388320.

[20] Carolee Stewart-Gardiner. "Using peer led team learning to assist in retention in computer science classes". In: *J. Comput. Sci. Coll.* 25.3 (Jan. 2010), pp. 164–171. ISSN: 1937-4771.

[21] National Center for Women and Information Technology. *Higher Ed Learning Circles*. https://ncwit.org/program/highered-programs/extension-services/extension-services-learning-circles/.

Exploring Faculty and Student Perspectives on GenAI in Higher Education[*]

Thad Crews, John Erickson, and Tong Wu
Analytics and Information Systems Department
Western Kentucky University
Bowling Green, KY 42101
`{thad.crewsii,john.erickson,tong.wu}@wku.edu`

Abstract

This study explores the growing impact of GenAI tools in higher education. The study involves a repeated cross-sectional survey of faculty and students to identify valuable insights into evolving patterns and preferences regarding the impact of ChatGPT and other AI tools in higher education. Results are reported with insights for faculty and policy makers.

1   Introduction

Generative AI (GenAI) [18] has become widely available and often controversial tool in higher education. As a language model, ChatGPT can pass the SATs and the bar exam [15]. And ChatGPT writes very well. One study compared written responses from physicians and those from ChatGPT to real-world health questions, and a panel of licensed healthcare professionals preferred ChatGPT's responses 79% of the time, rating them higher quality and more empathetic [3].

There is no doubt that ChatGPT and other AI tools will significantly impact teaching and learning. Generative AI (GenAI) can enhance learning experiences by improving academic writing, fostering creativity, and supporting

the development of innovative projects [6, 12, 21]. Additionally, GenAI has been found to boost students' coding abilities through quick feedback, which also enhances problem-solving, critical thinking, and coding confidence [22]. Faculty can benefit significantly from GenAI's ability to design and support customized learning experiences tailored to individual student needs [4, 9]. These tools have the potential to revolutionize teaching by providing personalized education, robust training support, and reducing grading time [17, 13]. Furthermore, GenAI can assist faculty in research by generating ideas, analyzing data, and aiding in writing, thereby increasing efficiency in the publication process [8]. On May 30, 2024, OpenAI announced **ChatGPT Edu**, powered by GPT-4o, as "a version of ChatGPT built for universities to responsibly deploy AI to students, faculty, researchers, and campus operations."[14]

Despite its benefits, the integration of GenAI in higher education comes with challenges. Previous studies have highlighted issues such as biases inherent in AI systems [23], ethical considerations [20, 24], risks of plagiarism [10, 16], and broader concerns about academic integrity [11]. These challenges underscore the need for responsible implementation and careful consideration of GenAI's limitations [6].

GenAI holds considerable potential as a valuable tool for both students and faculty in higher education. However, addressing associated challenges is crucial for its successful and ethical integration into educational practices. This study aims to identify some policies and insights to help with the transition of GenAI technology in the most beneficial manner for faculty and students.

2   Related Work

In 1984, Benjamin Bloom described the "The 2 Sigma Problem" [5] which remains one of the most optimistic outcomes regarding the possible impact of technology for improved teaching and learning. Figure 1 shows the two-sigma shift from conventional instruction to expert instruction to 1-to-1 personal instruction. Bloom's idealistic goal was for technology to one day facilitate that highest learning context of 1-to-1 personal instruction.

Another example of technology's impact on teaching and learning is the growing use of flipped classroom design, an instructional approach where students learn new material outside of class, typically through pre-recorded videos or other online resources, and then use class time to apply, practice, and reinforce that knowledge through active learning activities such as discussion, group work, and problem-solving. This model reverses flips the classroom from a lecture space to active learning space, allowing for personalized interactions during class and self-paced learning at home [1, 2]

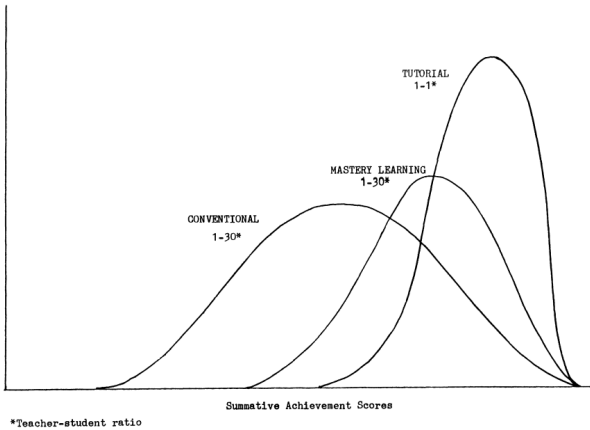Active learning is generally understood as being the more effective than

Figure 1: Bloom's 2 Sigma Problem

passive learning. The active "role" can include the student, the teacher, or the learning tool. Textbooks are excellent tools, but they are not active tools, they cannot customize their interaction based on the individual student. Technology can be used as an "Active Tool" to support 3D Active Learning [7] as shown in Figure 2 below. Online and hybrid classes can particularly benefit from having active technology engagement outside the classroom.
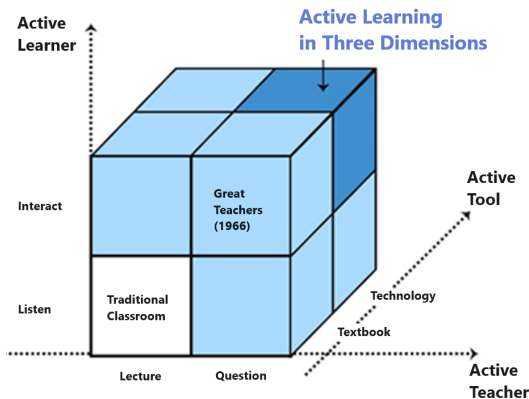

Figure 2: 3D Active Learning

There is no question that ChatGPT and other AI tools will significantly

impact our teaching and learning. This study attempts to identify some of the best policies and insights to help with the transition of GenAI technology in the most beneficial manner for faculty and students.

3   Methodology

For this research, we used a web-based survey to examine how higher education is being impacted by ChatGPT and other AI tools. The survey included determining faculty and student awareness regarding these tools, their use of the tools, and their perceived impact (positive and negative) on teaching and learning. The survey was administered in fall 2023 (12 months after the public release of ChatGPT 3.5) and again in spring 2024 (18 months after the 3.5 release). The survey was administered through Qualtrics Manager services and made available to external higher education faculty and students. Having repeated cross-sectional data provides valuable insights into evolving patterns and preferences.

4   Results and Discussion

In fall 2023, the authors received 216 total responses, composed of 105 faculty and 111 students. For the spring 2024 collection, there were 207 responses with 104 faculty and 103 students.

4.1   Adoption Rate

Figure 3 below shows the fall 2023 adoption rate of 61.3% for students and 68.6% for faculty. The adoption rates were even higher for both groups in spring 2024, 66.7% for student and 80.0% for faculty. These numbers are already into the "Late Majority" and even "Laggards" according to the traditional technology adoption curve [19]. These individuals are traditionally not as comfortable with rapid technology change, but the current environment may be forcing them to respond more quickly than they prefer.

   The number of hours of use also increased. In response to the question "In the last 30 days, how many hours did you spend using GenAI tools (such as ChatGPT, Bard, Dall-E2, etc.)?" the mean response was in 15.6 hours in F23 and 20.9 hours in S24. These adoption rate for for GenAI tools is significantly faster than traditional technology adoptions, which is creating challenges for faculty and administrators trying to manage the smooth transition of the technology into the teaching and learning experience.

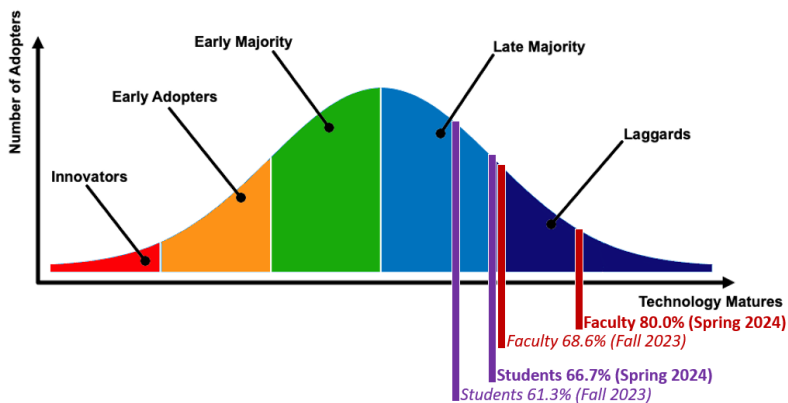How often do you use generative artificial intelligence (GenAI) tools such as ChatGPT, Bard, and Dall-E2?

Figure 3: GenAI reported usage rates for faculty and students

4.2 Beliefs on impact on student learning

The survey results show the majority of faculty and students believe GenAI will have a positive impact on student learning (Figure 4). Students belief in positive impact is 55% and 57% across the to survey periods. Faculty were even more optimistic in their belief of positive impact 65% and 67%.

"AI-users" were determined from the survey question: "Which of the following best describes your own use of generative AI writing tools (e.g., Chat-GPT)?" with *AI-user defined as monthly, weekly, or daily usage of generative AI writing tools. For faculty and students identified as AI-users, the positive impact belief is even higher (71% for students and 78% for faculty). This is interesting because many technology adoptions have failed to live up to the hype. With GenAI technology, however, the users of the tools are even more optimistic about their positive role in the teaching and learning experience.

4.3 Beliefs on student need to know for success in professional settings

There is a frequent tension in higher education about providing an education that prepares students for career success after graduation versus providing an education valued for its intrinsic worth. Without taking a side on that debate, we asked the question "Do students need to know how to effectively use generative AI tools to succeed in professional settings?" Once again the majority
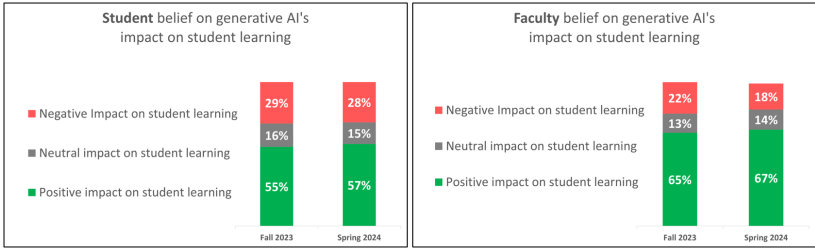
Figure 4: Beliefs about GenAI's impact on student learning

of both faculty and students reported that students do need to know how to effectively use GenAI to succeed in professional environments (see Figure 5). And once again, the AI-users were even more supportive of the technology and its impact on students in their future professional lives.
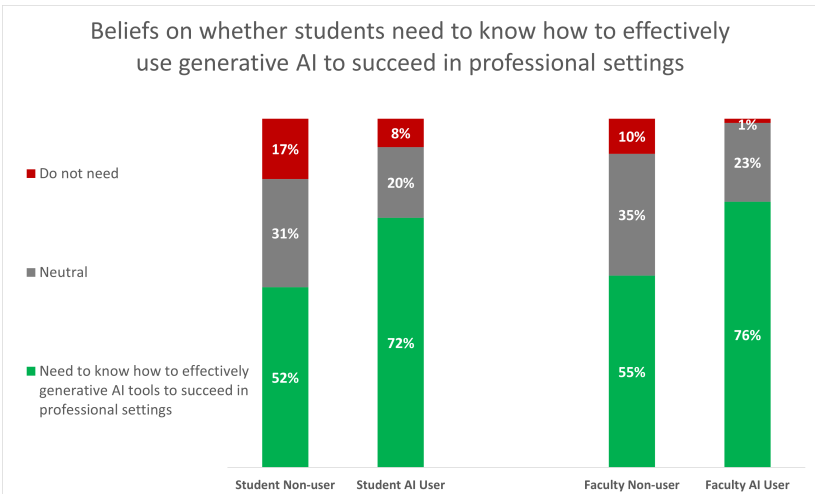

Figure 5: Beliefs about GenAI use in professional settings

4.4 Student likelihood to use GenAI technology even if the technology is prohibited

Figure 6 shows the responses to the question asked only of students: "(Student Only) If your instructor or institution prohibits the use of generative AI writing tools, how likely are you to still use something like ChatGPT?"

164

The responses were eye-opening, as most students (52% in F23, 50% in S24) indicated they were at least somewhat likely to still use the AI tools even if the use of those tools were prohibited. Even among non-users 1 in 5 were still open to using AI tools in the future even if their use were prohibited. For the students who are already AI-users, an even higher percentage (66% in F23, 62% in S24) said they were at least somewhat likely to continue using AI tools even if the tools are prohibited.
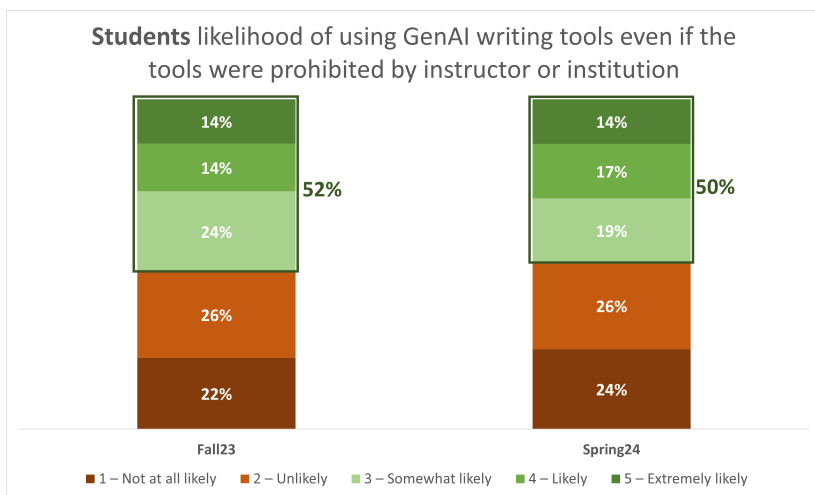

Figure 6: Students' likelihood of using GenAI tools even if prohibited

These student responses should be strongly considered by faculty and administrators when making plans and policies regarding student use of AI tools.

4.5   Preferred Syllabus Policy Statement

Students and Faculty were asked about their preferred syllabus policy statement regarding AI tool use. The four options were:

- **Full Use of AI Permitted:** In this class you are free to use artificial intelligence (AI) tools on assignments and activities in this course. If you chose to utilize AI, you will be expected to properly document and cite this information. Examples of citing AI are available at https://libguides.abc.edu/stylewrite/ai.

- **Limited Use of AI Permitted:** Artificial intelligence (AI) tools may be use for (list assignments or types of activities such as brainstorming or

gathering ideas) with appropriate citation, but not for (list assignments). If you are unsure if you are using AI tools appropriately in this course, I encourage you to visit with me. Examples of citing AI are available at https://libguides.abc.edu/stylewrite/ai.

- **Permission Required for Use of AI:** In general, I expect that the work you submit in this class will be your own and you are not authorized to use artificial intelligence (AI) tools. However, there may be specific assignments or activities in which we will utilize these tools to enhance your learning experience. In these instances, I will provide you with additional information about the assignment including instructions on how AI will be employed and cited. Again, unless permission is granted, you are expected to complete assignments without substantive assistance from others, including AI tools.

- **AI Tools Prohibited:** Artificial intelligence (AI) tools are not permitted for any type of work in this class. If you choose to use these tools, your actions will be considered academically dishonest and a violation of the University Student Code of Conduct.

The option "Limited Use of AI Permitted" was the most preferred option from both groups across both samples. Over a third of all students (44%) and all faculty (38%) selected this option in the most recent sampling (see Figure 7).

The option "Full Use of AI Permitted" took over second place in the most recent sampling with students at 19% and faculty at 33%. The student and faculty numbers both increased for this policy option.

The option "Permission Required for Use of AI" had a sharp 13% faculty drop in spring 2024, the only double-digit group change from the two samples. One possible explanation for the faculty drop is the amount of extra faculty work required with this syllabus policy. Another factor might be an unwanted notion that the faculty member is choosing the best of ALL relevant AI tools, which is probably not realistic. The double-digit faculty change on this option is significant and a stark contrast to the 1% student change for this item, the smallest of the eight changes.

The option "AI Tools Prohibited" was the least preferred option by both faculty and students in both fall 2023 and spring 2024. However, the responses were non-zero and increasing, indicating there may be situations based on design, content, or other factors where the best learning option might be a total absence of AI tools.
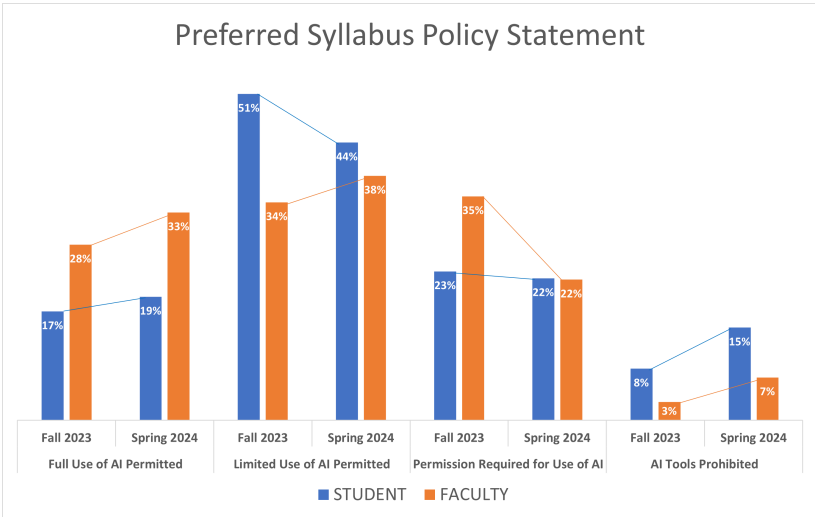
Figure 7: Policy Preferences

5 Discussion and Conclusion

The fall 2023 and spring 2024 national surveys of faculty and students provides multiple insights into the adoption and attitudes of GenAI technology use in higher education. Repeated cross-sectional data provides valuable insights into evolving patterns and preferences, providing valuable insights for faculty and policy makers.

One key insight is the rapid rate of adoption for GenAI technology. The adoption curve for technology traditionally involves a much longer timeline, allowing "late adopters" to observer the technology and its impact for a significant period of time before outside pressures force them to transition into the new technology [19]. The rapid adoption rate of GenAI helps explain the high-level of faculty interest (if not concern) surrounding this particular technology and its impact.

Another key finding is the high number of students who reported they are likely to continue to use ChatGPT even if instructors or institutions prohibited its use. Even if these hypothetical numbers are double the actual numbers in practice, it still should give significant pause to faculty as they make plans regarding the use of GenAI tools in their own courses.

Additional findings regarding perceived positive impact, professional impact and policies were also presented, with each instance suggesting that generative AI tools are going to have a long-term impact in higher education.

References

[1] L. Abeysekera and P. Dawson. "Motivation and cognitive load in the flipped classroom: definition, rationale and a call for research". In: *Higher education research and development* 34.1 (2015), pp. 1–14.

[2] G. Akcayir and M. Akcayir. "The flipped classroom: A review of its advantages and challenges". In: *Computers and Education* 126 (2018), pp. 334–345.

[3] J. Ayers and D. Smith. "Comparing Physician and Artificial Intelligence Chatbot Responses to Patient Questions Posted to a Public Social Media Forum". In: *JAMA Internal Medicine* 183.6 (2023), pp. 589–596.

[4] D. Baidoo-Anu and L. Ansah. "Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning". In: *Journal of AI* 7.1 (2023), pp. 52–62.

[5] B. Bloom. "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring". In: *Educational researcher* 13.6 (1984), pp. 4–16.

[6] C. Chan and W. Hu. "Students' voices on generative AI: Perceptions, benefits, and challenges in higher education". In: *International Journal of Educational Technology in Higher Education* 20.1 (2023), pp. 20–43.

[7] T. Crews. "Active Learning Across Three Dimensions: Integrating Classic Learning Theory with Modern Instructional Technology". In: *International Journal of Learning, Teaching and Educational Research* 16.1 (2017), pp. 72–83.

[8] EAM van Dis et al. "ChatGPT: five priorities for research". In: *Nature* 614.7947 (2023), pp. 224–226.

[9] L. Kohnke, B. Moorhouse, and D. Zou. "Exploring generative artificial intelligence preparedness among university language instructors: A case study". In: *Computers and Education: Artificial Intelligence* 5 (2023). DOI: `10.1016/j.caeai.2023.100156`.

[10] W. Lim et al. "Generative AI and the future of education: Ragnarök or reformation? A paradoxical perspective from management educators". In: *The international journal of management education* 21.2 (2023). DOI: `10.1016/j.ijme.2023.100790`.

[11] C. Lo. "What is the impact of ChatGPT on education? A rapid review of the literature". In: *Education Sciences* 13.4 (2023). DOI: `10.3390/educsci13040410`.

[12] A. Malik, Y. Pratiwi, and A. Darwis. "Exploring artificial intelligence in academic essay: Higher education students perspective". In: *International Journal of Educational Research Open* 5 (2023). DOI: `10.1016/j.ijedro.2023.100296`.

[13] A. Mizumoto and M. Eguchi. "Exploring the potential of using an AI language model for automated essay scoring". In: *Research Methods in Applied Linguistics* 2.2 (2023). DOI: `10.1016/j.rmal.2023.100050`.

[14] OpenAI. *https://openai.com/index/introducing-chatgpt-edu/ (referenced May 30, 2024)*.

[15] OpenAI. *openai.com/research/gpt-4 (referenced May 3, 2023)*.

[16] R. Peres et al. "On ChatGPT and beyond: How generative artificial intelligence may affect research, teaching, and practice". In: *International Journal of Research in Marketing* 40.2 (2023), pp. 269–275.

[17] A. Stojanov. "Learning with ChatGPT 3.5 as a more knowledgeable other: An autoethnographic study". In: *International Journal of Educational Technology in Higher Education* 20.1 (2023), pp. 20–35.

[18] C. Stokel-Walker and R. Van Noorden. "What ChatGPT and generative AI mean for science". In: *Nature* 614.7947 (2023), pp. 214–216.

[19] E. Straub. "Understanding technology adoption: Theory and future directions for informal learning". In: *Review of educational research* 79.2 (2009), pp. 625–649.

[20] A. Tlili et al. "What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education". In: *Smart learning environments* 10.1 (2023). DOI: `10.1186/s40561-023-00237-x`.

[21] D. Yan. "Impact of ChatGPT on learners in a L2 writing practicum: An exploratory investigation". In: *Education and Information Technologies* 28.11 (2023), pp. 13943–13967.

[22] R. Yilmaz and F. Yilmaz. "Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning". In: *Computers in Human Behavior: Artificial Humans* 1.100005 (2023).

[23] A. Yusuf, N. Pervin, and M. Román-González. "Generative AI and the future of higher education: a threat to academic integrity or reformation? Evidence from multicultural perspectives". In: *International Journal of Educational Technology in Higher Education* 21.21 (2024). DOI: `10.1186/s41239-024-00453-6`.

[24] W. Zhu et al. "Could AI ethical anxiety, perceived ethical risks and ethical awareness about AI influence university students' use of generative AI products? An ethical perspective". In: *International Journal of Human–Computer Interaction* (2024), pp. 1–23. DOI: 10.1080/10447318.2024.2323277.

Reviewers — 2024 CCSC Southeastern Conference

Achee, Dr. Bonnie ....Southeastern Louisiana University, Hammond, LA
Ali, Farha ....Lander University, Greenwood, SC
Alvin, Chris ....Furman University, Greenville, SC
Bennett, Brian .... East Tennessee State University, Johnson City, TN
Besmer, Andrew .... Winthrop University, Rock Hill, SC
Burton, Sharon ....Capitol Technology University, Laurel, MD
Crews, Thad .... Western Kentucky University, Bowling Green, KY
D'Antonio, Lawrence ....Ramapo College, Mahwah, NJ
Dasgupta, Anurag ....Valdosta State University, Valdosta, GA
Demirhan, Hilmi University of North Carolina - Wilmington, Wilmington, NC
Digh, Andy .... Mercer University, Macon, GA
Dogan, Gulustan ...University of North Carolina - Wilmington, Wilmington, NC
Drucker, Rebecca ....Furman University, Greenville, SC
Ferreras, Jennifer ....Marymount University, Arlington, VA
Gesick, Richard ....Kennesaw State University, Kennesaw, GA
Goodard, Wayne ....Clemson University, Clemson, SC
Hills, Mark Alan ....Appalachian State University, Boone, NC
Holliday, Mark ....Western Carolina University, Cullowhee, NC
Hong, Gongbing .... Georgia College and State University, Milledgeville, GA
Hutchings, Dugald Ralph ....Elon University, Elon, NC
Lewis, Adam ....Athens State University, Athens, AL
Li, Rao ....University of South Carolina - Aiken, Aiken, SC
Lindoo, Edward .... Regis University, Stuart, FL
Liu, Yi .... Georgia College & State University, Milledgeville, GA
Matfield, Ryan Stephen ....Elon University, Elon, NC
Rhujittawiawat, Theppatorn ....Claflin University, Orangeburg, SC
Roach, Jeff .... East Tennessee State University, Johnson City, TN
Sadeghpour, Shadi .... The Citadel, Charleston, SC
Shende, Anil .... Roanoke College, Salem, VA
Spurlock, Scott ....Elon University, Elon, NC
Sriram, Ramaier ....Claflin University, Orangeburg, SC
Treu, Kevin .... Furman University, Greenville, SC
Verdicchio, Michael .... The Citadel, Charleston, SC
Works, Karen ....Florida State University, Panama City, FL
Yao, JF ....Georgia College & State University, Milledgeville, GA