The Journal of Computing Sciences in Colleges

Papers of the 25th Annual CCSC Northwestern Conference

October 13th-14th, 2023
Eastern Washington University
Spokane, WA

Baochuan Lu, Editor
Southwest Baptist University

Sharon Tuttle, Regional Editor
Cal Poly Humboldt

Volume 39, Number 1

October 2023

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

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

CCSC National Partners

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

Gold Level Partner
Rephactor
ACM2Y
ACM CCECC

Welcome to the 2023 CCSC Northwestern Conference Eastern Washington University

On behalf of the Conference Steering Committee and Northwest Regional Board, I welcome you to the Twenty Third Annual Consortium for Computing Sciences (CCSC) Northwestern Regional Conference, held at the Eastern Washington University Catalyst Building. We are very excited to meet you all in person.

Many individuals and groups helped coordinate and support this year's conference and I want to thank them for all their time and effort. This year's conference includes 11 papers, five panels/tutorials, and the keynote. All papers, panels, and tutorials went through the regular peer review process. The steering committee accepted 11 out of 16 papers (69% acceptance rate). We had colleagues across the region serve as professional reviewers and we recognize their generous efforts in providing time and guidance in the selection of our conference program.

Our Keynote speaker, Graham Moorehead will talk about what is thought and how is human thought different from artificial intelligence thought.

Of course, none of this would be possible without the support of our national and local partners. A final thank you goes out to you the attendees whose participation is essential not only to the continuance of conferences such as this, but also for the continued communication and collegiality you provide between all of us involved in the advancement and promotion of our discipline.

We are excited to invite you to our campus, and hope you enjoy the conference and the chance to interact with your colleagues at our annual gathering.

Stu Steiner
Eastern Washington University
CCSC-NW 2023 Conference Chair

2023 CCSC Northwestern Conference Steering Committee

Stuart Steiner, Conference Chair/Site Chair/Speakers Chair	Eastern Washington University
Haiyan Cheng, Program Chair	Willamette University
Ben Tribelhorn, Papers Chair	University of Portland
Markus Geissler, Panels & Tutorials	Cosumnes River College
Daniel Conte de Leon, Partners Chair	University of Idaho
Calvin Deutschbein, Student Posters Chair	Willamette University

Regional Board — 2023 CCSC Northwestern Region

Shereen Khoja, Regional Representative	Pacific University
Peter Drake, Registrar	Lewis Clark University
Dan Ford, Treasurer	Linfield College
Sharon Tuttle, Editor	Cal Poly Humboldt University
Gayathri Iyer, Past Conf. Chair	Portland Community College
Stuart Steiner, Next Conf. Chair	Eastern Washington University
Razvan Mezei, Webmaster	Saint Martin's University

The Shape of Thought[*]

Keynote

Graham Moorehead

CEO - Pangeon

Computer Science Department
Gonzaga University
Spokane, WA 00528

The study of artificial intelligence forces us to understand our own. What is thought – how is human thought different from artificial thought?

Graham Morehead is a professor teaching AI at Gonzaga University, and the CEO of Pangeon. For over 25 years, Graham has been developing AI and machine learning solutions to difficult problems. His research led to several TEDx talks and several tech companies. He is currently working to solve problems related to wildfire, real estate, and linguistics.

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

Position Paper on Teaching Operating Systems using the Rust Programming Language*

Bryan Dixon
Computer Science Department
California State University - Chico
Chico CA, 95929
bcdixon@csuchico.edu

Abstract

In the past few years, the Rust Programming Language has quickly gained adoption in a myriad of applications. This adoption, especially into the Linux Kernel, is a clear motivation that there may be a reason to have students learn Rust during their computer science studies. This position paper will discuss a case for why instructors should incorporate Rust as part of an operating systems course and how one would accomplish this.

1 Introduction

The Rust programming language publicly appeared in 2010 and has been a stable release since 2014[19]. Since its stable release, it has been adopted by quite a few high-profile companies, including Amazon, Facebook (Meta), Google (Alphabet), and Microsoft. The popularity of Rust has also exploded since its release, as it has been the most loved language in Stack Overflow's annual Developer Survey for the past seven years[21]. Additionally, that survey found Rust to be the most wanted technology, indicating that developers have expressed interest in developing with Rust. Other surveys have listed Rust as

*Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

one of the top twenty most popular programming languages, including those from IEEE Spectrum, Redmonk, and Tiobe[5][1][22].

An essential feature of Rust is that it is a memory-safe programming language that is provably safe[10]. In lower-level development, security is paramount, so the memory safety aspect of Rust makes it an attractive choice of language to move forward on for teaching operating systems. Another critical feature of Rust is that it has fast and predictable performance. Rust has no garbage collector, so there are no garbage collection performance losses, as with many other general-purpose programming languages. Instead, Rust solves the problem with features inspired by functional programming languages.

The promise of Rust as a safer performant systems language has led to widespread adoption by quite a few high-profile companies and, more importantly, adoption into the Linux Kernel itself. This adoption started originally with Amazon developing containers and system code for their custom cloud kernels[24]. Amazon made a clear case for Rust, arguing how Rust is nearly as efficient and performant as C but has added memory protections that C does not provide. Also noted was a correlation between performance and energy consumption. Amazon compared the energy usage and performance of 27 different languages and found Rust to be one of the top performers in terms of energy efficiency[16].

In September 2022, the framework for future Linux Kernel development was introduced, highlighting efforts to allow Rust modules in the kernel due to Rust's memory safety and the potential corresponding stability improvements[18]. Furthermore, Rust started getting committed to the Linux Kernel, starting with version 6.1 [23][25][2]. A recent development is a consideration for GNU GCC Compiler to add support for Rust, allowing Rust to benefit from the entire GNU ecosystem and plug-ins for GCC developed for over 35 years[6].

2 Rust in Education

Outside of Rust forums and other discussion boards like Quora, there is no evidence schools are starting to swap to teaching introduction to programming courses in Rust[20][15]. The learning curve for Rust can be more challenging than that of C or C++, so this may be the reason for the absence of adoption in introductory programming courses. However, there could still be benefits to incorporating Rust into upper-division programming courses.

2.1 Rust for Systems Education

There are several examples of both systems programming courses and graduate-level operating systems course projects that have explored the feasibility of applying Rust in their courses[9][14][11][12][13][3]. These Rust use cases are all graduate-level Rust systems programming projects that one could expect to see at Ph.D. granting institutions for their graduate operating systems courses. I have taught our graduate operating systems course numerous times for M.S. students, and they would struggle with these deep-dive projects.

Before Rust was even stable, a professor at the University of Virginia taught an undergraduate operating systems course entirely in Rust[8]. They discussed their Rust-based assignments along with the motivations and flaws of this approach. The two primary reasons they argued against teaching operating systems in Rust were its immaturity and learning curve. The entire course in Rust has shortcomings since so much of the Linux kernel is still based on C. The C programming language is still the most popular programming language. With the power of the C Foreign Function Interface (CFFI), teaching C alongside a more modern alternative such as Rust is likely a stronger approach[4].

3 Teaching Operating Systems with Rust

There are two significant hurdles to teaching an operating systems course in Rust. The obvious one is that students are unlikely to have learned Rust in introductory courses, so the course will need to include time for teaching students the basics of Rust. The second is having assignments that balance teaching operating systems fundamental knowledge and the benefits of using Rust for systems programming.

3.1 Rust Introduction Hurdle

Some institutions expect courses to have at least one assignment due before the add or drop date to help students determine whether to stay in or drop a course. My institution's add/drop date is at the end of the fourth week of the semester. This time window makes creating substantive assignments regarding operating system concepts and fundamentals difficult without students feeling overwhelmed. I have solved this problem with a simple kernel module authoring assignment that students can start on day one of the course without any additional information. I could replace this kernel module assignment with lab time introducing students to Rust and getting them started with Rust programming assignments.

Rust has several novel language features that must be covered in an introductory set of labs or assignments. Introducing students to these novel features would be necessary for using Rust and setting students up for success using Rust in systems programming assignments later in the course. Here is a list of key topics that should likely be covered:

- Variables and Mutability
- Data types
- Ownership
- Structs
- Error Handling
- Crates

Numerous other common programming topics exist, like how Rust handles loops, functions, and other features. However, the most significant bits that students we have introduced Rust to have gotten tripped up on are mutability and ownership. Labs may also need additional time regarding Rust-specific threading and synchronization tools if these are not covered in lectures. Such tools include covering Rust crates regarding concurrent programming tools not included in standard Rust like Crossbeam [7].

Some students would benefit from learning Rust through instructional videos that go beyond reading the Rust documentation alone. The benefit of this approach is that lecture time does not need to be dedicated to Rust instruction, leveraging lab time instead. Institutions that do not have operating systems courses paired with labs could instead have Rust introduction videos available that students could watch outside lecture time. Developing such tailored Rust introduction videos would be necessary to ensure the adoption of such a proposed teaching of OS to a broader set of schools.

3.2 Rust Assignments

The need for some C programming is due to the expectation and need for students to come out of our operating systems courses knowing pthreads. With how our operating systems course fits into the curriculum, we could not move the course to 100 percent Rust. Nevertheless, I could see a pathway for all the other hands-on assignments in my operating systems course to become Rust-based. I already offer extra credit for students to re-implement the pthreads assignment with Rust.

Specifically, these libraries would be extremely helpful in implementing such assignments:

- nix::unistd - wrapper for standard nix system calls including fork, exec, setpgid, pipe, sleep, and others
- nix::sys::wait - wrapper for wait, waidpid nix system calls
- signal_hook - Library for easier and safe Unix signal handling
- crossbeam - external library with multi-producer, multi-consumer channels, and other tools for concurrent programming
- std::sync - standard library with mutexes, conditional variables, and the rust Atomically Reference Counted (Arc) struct.

I give students hands-on experience with virtual paging and scheduling through assignments where students must implement libraries tested via a simulation. These kinds of assignments would be easier to migrate to Rust; however, developing the simulator to be just as robust and still have the same learning outcomes would be a potential challenge.

Some difficulties are inherent in rebuilding these types of assignments from C into Rust in a way that still makes sense and does not add significant hurdles to student success. Getting time to recreate these assignments may necessitate someone obtaining NSF funding from an Improving Undergraduate STEM Education (IUSE) grant like Saverio Perugini did to re-envision a modern operating systems course[17]. The time is necessary since these are such complex and problematic programs to write while ensuring consistency in their behaviors.

4 Conclusion

There is a clear case for moving toward Rust-based development for modern operating systems. Future students would benefit significantly from an operating systems course emphasizing Rust development that still covers some core C functionality. If any such course of materials comes to fruition, it would also be helpful to see it published as a Course in a Box to make it easier for other instructors to implement the course independently. The only way we are likely to see faculty find time to put together such a course is with grant funding or a sabbatical due to the difficulty of putting together these assignments in a polished state.

References

[1] Stephen O'Grady |@sogrady |March 28, Stephen O'Grady, and Name *. *The Redmonk Programming Language Rankings: January 2022*. `https://redmonk.com/sogrady/2022/03/28/language-rankings-1-22/`, last accessed on 2022-11-19. Mar. 2022.

[2] Arindam. *Linux Kernel 6.3 Rc1 Released with Intel VPU Driver, Rust Updates*. `https://www.debugpoint.com/linux-kernel-6-3-rc1/`, last accessed on 2023-4-4. Mar. 2023.

[3] Abhiram Balasubramanian et al. "System programming in rust: Beyond safety". In: *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. 2017, pp. 156–161.

[4] Tiemoko Ballo and Alex James. "Can't Spell "Curriculum" Without "C"". In: *Rust Education Workshop* (2022).

[5] Stephen Cass. *Top programming languages 2022*. `https://spectrum.ieee.org/top-programming-languages-2022`, last accessed on 2022-11-19. Sept. 2022.

[6] David Cassel. *Rust Support Is Being Built into the GNU GCC Compiler*. `https://thenewstack.io/rust-support-is-being-built-into-the-gnu-gcc-compiler/`, last accessed on 2023-4-4. Mar. 2023.

[7] *Crossbeam - Tools for concurrent programming*. `https://crates.io/crates/crossbeam`, last accessed on 2023-4-4.

[8] David Evans. *Using rust for an undergraduate os course*. `https://rust-class.org/pages/using-rust-for-an-undergraduate-os-course.html`, last accessed on 2022-11-19. 2013.

[9] Gabriel Ferrer. "The Pluggable Interrupt OS: Writing a Kernel in Rust". In: (2022).

[10] Ralf Jung. "Understanding and Evolving the Rust Programming Language". In: (2020).

[11] Stefan Lankes, Jens Breitbart, and Simon Pickartz. "Exploring rust for unikernel development". In: *Proceedings of the 10th Workshop on Programming Languages and Operating Systems*. 2019, pp. 8–15.

[12] Stefan Lankes et al. "RustyHermit: A Scalable, Rust-Based Virtual Execution Environment". In: *International Conference on High Performance Computing*. Springer. 2020, pp. 331–342.

[13] Amit Levy et al. "The case for writing a kernel in rust". In: *Proceedings of the 8th Asia-Pacific Workshop on Systems*. 2017, pp. 1–7.

[14] Yuanzhi Liang et al. “Rustpi: A Rust-powered Reliable Micro-kernel Operating System”. In: *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2021, pp. 272–273.

[15] Dustin Matlock. *Learning rust as a first programming language.* `https://users.rust-lang.org/t/learning-rust-as-a-first-programming-language/12919`, last accessed on 2022-11-19. Sept. 2017.

[16] Shane Miller and Carl Lerche. *Sustainability with Rust.* `https://aws.amazon.com/blogs/opensource/sustainability-with-rust/`, last accessed on 2022-11-19. Feb. 2022.

[17] Saverio Perugini and David J Wright. “Developing a contemporary operating systems course”. In: *Journal of Computing Sciences in Colleges* 34.1 (2018), pp. 155–156.

[18] Liam Proven. *Rust is coming to the linux kernel.* `https://www.theregister.com/2022/09/16/rust_in_the_linux_kernel/`, last accessed on 2022-11-19. Sept. 2022.

[19] *Rust.* `https://www.rust-lang.org/`, last accessed on 2022-11-19.

[20] *Should I learn rust as a first programming language?* `https://www.quora.com/Should-I-learn-Rust-as-a-first-programming-language`, last accessed on 2022-11-19.

[21] *Stack overflow developer survey 2022.* `https://survey.stackoverflow.co/2022/?utm_source=so-owned&amp;utm_medium=announcement-banner&amp;utm_campaign=dev-survey-2022&amp;utm_content=results#section-most-loved-dreaded-and-wanted-programming-scripting-and-markup-languages`, last accessed on 2022-11-19. 2022.

[22] *Tiobe index.* `https://www.tiobe.com/tiobe-index/`, last accessed on 2022-11-19. June 2022.

[23] Linux Torvalds. *Merge tag 'rust-v6.1-rc1' of https://github.com/Rust-for-Linux/linux.* `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=8aebac82933ff1a7c8eede18cab11e1115e2062b`, last accessed on 2022-11-19. Oct. 2022.

[24] Steven Vaughan-Nichols. *Aws Introduces Bottlerocket: A rust language-oriented linux for containers.* `https://www.zdnet.com/article/aws-introduces-bottlerocket-a-rust-language-oriented-linux-for-containers/`, last accessed on 2022-11-19. Sept. 2020.

[25] Steven J. Vaughan-Nichols. *Rust in the linux kernel.* `https://thenewstack.io/rust-in-the-linux-kernel/`, last accessed on 2022-11-19. Oct. 2022.

AI and Society: Teaching AI to non-STEM Students[*]

Pejman Khadivi
Computer Science Department
Seattle University
Seattle, WA 98122
khadivip@seattleu.edu

Abstract

Artificial Intelligence (AI) is a booming technology with a broad range of applications in almost any aspect of life, technology, science, and society. However, despite its popularity, AI education to non-STEM students is not addressed properly. In this paper, we introduce a new AI course, named AI and Society, that has been designed for non-STEM students with very limited knowledge about math and computer science. In this course, the ultimate goal is to review fundamental concepts of AI and study the application of AI in major global issues such as healthcare, sustainability, transportation, and digital security. Furthermore, the course covers ethical issues in AI, and discuss different ethical aspects of AI systems such as algorithm and technology bias, accountability, and safety.

1 Introduction

Over the past decade, artificial intelligence (AI) has been changed to a ubiquitous technology. AI is now a universally accepted approach to tackle a wide range of problems with different levels of complexity. With recent advancements in computer science and engineering, data collection technologies, robotics, and telecommunications, AI is going to play a crucial role in a wide

[*]Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

range of applications from digital security to healthcare. However, while AI is a ubiquitous technology, students from different fields of study do not receive appropriate training about AI. Hence, after graduation, their expectations are not necessarily reasonable, and they may suffer from lack of knowledge about AI, its capabilities, and the corresponding ethical issues.

Due to the necessity of public education of artificial intelligence, in recent years there has been efforts to develop curriculum for teaching AI to those with no background in computer science and math [1], [4], [6]. Akram et. al in [1] developed a curriculum by combining four major concepts in AI, including search, knowledge-representations systems, machine learning, and natural language processing (NLP). This curriculum is designed to teach AI in middle-grade classes. In [4] a visualization tool is introduced for exploration of word embedding in NLP. This interactive tool is designed to facilitate teaching AI in K-12 programs. The issues of teaching AI in K-12 is also addressed in [6], where authors focused on AI knowledge, skills, and attitude of students, using the help of lectures, teachers, and workbooks.

In [2] a game has been proposed to introduce generative adversarial networks to middle school students. Norouzi et al. are targeting high school students in [9], where they proposed a curriculum to teach machine learning in a one-month period. The course is designed around NLP. The authors then asked students to submit surveys and analysed the collected responses. Teaching machine learning to K-12 students is also explored in [7].

Teaching machine learning to non-STEM students has been addressed in [5]. In this paper, Garcia-Algarra mainly talks about the contents of a machine learning course that has been designed for non-STEM students at Comillas University, in Madrid, Spain. A brief overview of a course syllabus for teaching machine learning to non-technical students is reviewed in [14]. Long and Magerko have analyzed AI literacy in [8] in order to design learner-centered AI courses. In this paper, the major goal of the authors was to coming up with a definition for AI literacy, by reviewing the interdisciplinary literacy and extracting AI literacy competencies.

In this paper, we introduce a new course which has been designed for teaching to non-STEM students. In this course, the ultimate goal is to review fundamental concepts of AI and to study the application of AI in major global issues such as healthcare, sustainability, transportation, and digital security. Moreover, this course addresses the ethical issues in AI. This course is designed for students with limited background in mathematics, computer programming, and science. Hence, it is designed so that the course material does not rely on math and computer programming. Instead, the emphasis will be on demystifying the algorithmic thinking behind various AI algorithms and models and discussing the application of various approaches on real-world problems.

2 Course Requirements

At Seattle University, every undergraduate student needs to complete a sequence of *university core* (UCOR) courses. As the university website mentions [13], UCOR courses, beside other learning outcomes, are designed to help the students to *"develop academic skills such as writing, critical thinking, and analysis of information"*[13], and to *"becoming a strong interdisciplinary thinkers"*[13]. To get their BS/BA degrees, students need to complete 180 credits at Seattle University. The UCOR curriculum is a 60-credit program, distributed over 12 courses and in three modules:

- Module I: Engaging Academic Inquiry: which contains courses on writing, quantitative reasoning, creative interpretation, and inquiry seminars.
- Module II: Engaging Jesuit Traditions: which contains courses on theological exploration, philosophy, and ethics.
- Module III: Engaging the World: which contain courses on religion in a global context, and global challenges.

The *AI and Society* course is a *natural sciences and global challenges* course which falls under Module III. STEM students are not eligible to register for this course and hence, all the students in the class are non-STEM. Therefore, one of the major assumptions in this course is that the students do not have a strong mathematics background. Furthermore, the assumption is that the students are not familiar with computer programming.

Students in a *natural sciences and global challenges* course need to complete a project or write a term paper, as one of the core learning outcomes is to assist the students to be effective writers. Moreover, the students need to work on some global issues in an interdisciplinary setting, from a scientific perspective. Therefore, having a term project related to a global issue is a major requirement in these courses.

2.1 Students Sample Distribution

AI and Society is designed for non-STEM students. Therefore, students that register for the course are coming from non-STEM departments and colleges. As an example, in one offering of the course, 39% of the students were from *Albers School of Business and Economics* (with majors such as finance, marketing, and management), 48% of them were from *College of Arts and Sciences* (with majors such as visual arts, psychology, and creative writing) , and the remaining 13% were from other colleges (with majors such as organizational leadership, and digital technology).

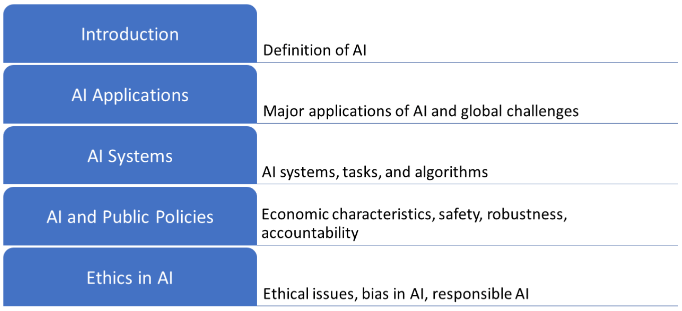

Introduction
Definition of AI
AI Applications
Major applications of AI and global challenges
AI Systems
AI systems, tasks, and algorithms
AI and Public Policies
Economic characteristics, safety, robustness, accountability
Ethics in AI
Ethical issues, bias in AI, responsible AI

Figure 1: Course sections.

3 Course Syllabus

Course sections are illustrated in Fig. 1. As mentioned before, the course is designed for non-STEM students as a university core course. Hence, one of the main concerns of this course is to address global challenges from scientific perspective. Furthermore, the course is also supposed to address ethical issues of the AI technology. Moreover, students do not have strong mathematics background and are assumed that have no background in computer programming. While many AI algorithms rely on mathematical theories, there are important AI algorithms that either do not need any math or can be simplified so that the students with no math background can understand the concepts. Local search algorithms such as hill-climbing, and nature inspired algorithms such as artificial neural networks, ant colony optimization, and genetic algorithms are some examples. For the assignments, students will be asked to apply these algorithms on simple problems manually, which means that no programming experience is required for this course.

The course content is designed based on the material from [12], *Artificial Intelligence: A modern approach* and [10], *Artificial Intelligence in Society* published by OECD. In this section, we introduce different sections of the course. Breakdown of the topics in each section are shown in Table 1.

In the *Introduction* section, the main concern is the definition of AI. This section starts by defining *intelligence*. Then, the instructor talks about various definitions of AI. A brief history of AI, Turing test, and rational agents are also introduced in this section.

In the *applications* section, the instructor talks about various applications of AI that are related to some global challenges. In this section, applications of AI in transportation, agriculture, science, health, criminal justice, security, and financial systems are explored.

The *AI systems* section starts with a crash-course on graph theory, as its concepts are required in the rest of this section. Then, after talking about search algorithms, local search solutions, and nature inspired models (such as genetic algorithm), rule based and case based AI systems, are introduced. This section also covers logic, and machine learning models.

Section *AI and public policies* is dedicated to public policy considerations and economic characteristics of an AI system. We will also introduce *human centered AI* concepts to the students.

The last section of the course is *ethics in AI*, where ethical issues of AI systems are described. In this section, bias, the importance of human dignity, and *responsible AI* topics are described.

Table 1: Breakdown of topics for each section.

Section	Topics
Introduction	Intelligence and AI, Turing Test, Components of AI, AI History, AI Applications, Rational Agents, Task-Environment
AI Applications	General Applications, AI in transportation, agriculture, finance, health, marketing, science, criminal justice, and security
AI Systems	Graph theory, Search for the Solution, Local Search Algorithms (e.g., Hill Climbing), Nature Inspired Algorithms (e.g., Genetic Algorithms), Machine Learning, Knowledge Representation, Logic and Fuzzy Logic, Rule Based Systems, Case Based Reasoning
AI and Public Policies	Human Centered AI, Transparency, Accountability, Robustness, Security, Safety, Economic characteristics
Ethics in AI	Ethical issues, Bias, Human Dignity, Responsible AI

3.1 Learning Outcomes

The learning outcomes of the course are as follows:

- L1: Demonstrate understanding of basic artificial intelligence approaches to problem solving.
- L2: Use algorithms and appropriate representation techniques for solving problems requiring learning and reasoning.
- L3: Analyze ethical principles relevant to design and usage of various AI-based systems to address societal problems
- L4: Describe different applications of AI

4 Major Assignments

There are three categories of major assignments in this course, which are described in this section. These categories are as follows:

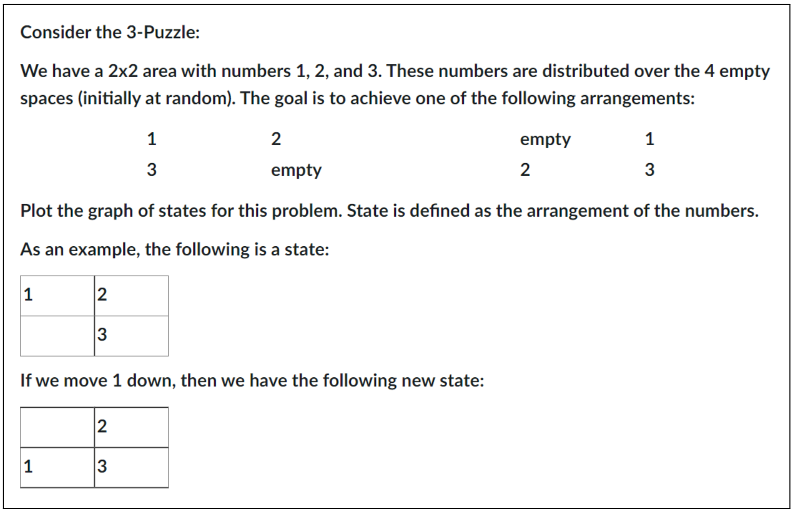
- Bi-weekly online discussions initiated by the instructor,
- Bi-weekly written assignments to demonstrate students' understanding of the concepts (e.g., manual tracing of an algorithm on a simple problem, or selecting an appropriate model for a problem), and
- Course project, which is a quarter-long activity.

Discussions are important assignments in this course as they let students to have an active participation in the class, share their thoughts with their peers, and use their prior knowledge to explore an AI question. Example discussion ideas are illustrated in Table 2. These discussion ideas may be used for in-class or online discussions.

Written assignments give the students the opportunity to practice on the concepts, models, and algorithms that have been discussed in the class. As an example, students may be asked to manually use the BFS algorithm to explore a graph, or to use cross-over operation in order to generate new chromosomes from two individuals in genetic algorithms. Three sample questions for written assignments are illustrated in Fig. 2. Since students are not supposed to have any programming background, no programming assignment is used in this course. In some discussions and written assignments, students are asked to either use an online application, or watch a video. As an example, as part of the ethics section, students are asked to complete a judgement on the *Moral Machine* activity [3]. We also asked them to watch the documentary *In the Age of AI* [11], on PBS as part of a discussion (Table 2).

Table 2: Some discussion ideas.

Title	Description
Daily example of AI	Think about one of your daily activities and recommend a way that AI may be used to improve your experience with that activity.
Self-Driving Cars Safety	What do you think about safety, regulations, and the corresponding ethical issues related to self-driving vehicles?
Age of AI	The documentary, In the Age of AI (Links to an external site.), talked about promises and challenges of AI. Pick one of the ethical issues that have been highlighted in the movie and express your own opinion about it.


Consider the 3-Puzzle:

We have a 2x2 area with numbers 1, 2, and 3. These numbers are distributed over the 4 empty spaces (initially at random). The goal is to achieve one of the following arrangements:

1	2	empty	1
3	empty	2	3

Plot the graph of states for this problem. State is defined as the arrangement of the numbers.

As an example, the following is a state:

1	2
	3

If we move 1 down, then we have the following new state:

	2
1	3

Figure 2: Sample written assignment question: A sample question about the concept of state and state graph.

4.1 Course Project

Students are required to complete a course project which addresses a global issue. The course project is interdisciplinary and students should be able to select a topic from a variety of topics, based on their background. During the course project, students are able to use knowledge they have learned in

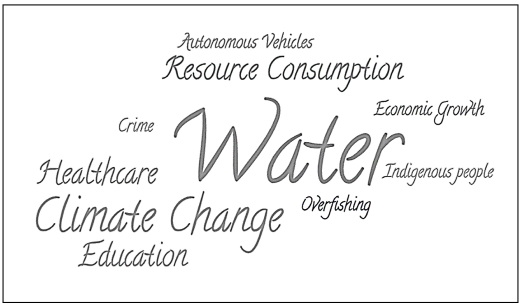


Figure 3: Sample course projects topics. Word sizes represent frequency of the topic in the class.

other courses, to solve an AI problem. Examples of course project areas are illustrated in Fig. 3. The students were supposed to select topics that represent global challenges. As Fig. 3 shows, the most popular topic was global water crisis, while application of AI in resource consumption, education, healthcare, and climate change were also frequently by the students.

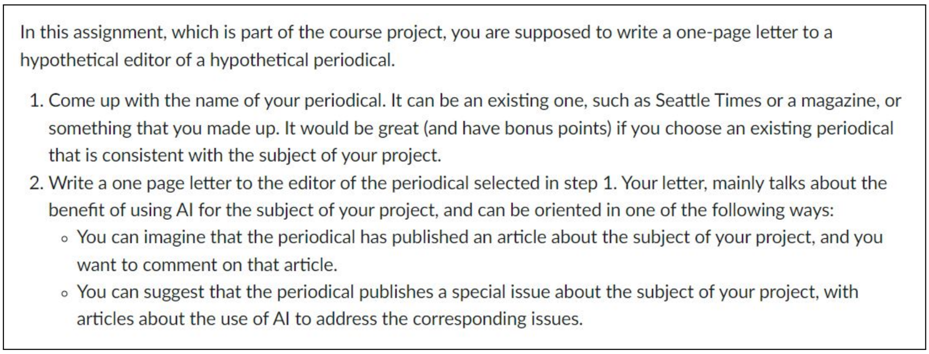
After doing the project, students will be required to practice technical communication in three mandatory ways, as follows:

1. Written practice: the students are required to write a "letter to the editor" of an appropriate magazine. Furthermore, the students have to write a term paper, reporting the project results and the corresponding literature review.
2. Oral practice: the students have to present their work to their classmates through short oral presentations.
3. Social media practice: the students have to come up with a series of significant number of meaningful and influential social media posts, regarding their work.

In addition to the term paper and presentation, each student will be required to submit a reflection questionnaire which not only asked the student to reflect on their work during the project but on how they have used their previous knowledge they have learned in other courses.

5 Students Performance and Feedback

The students reaction to the course was positive in general. Homework assignments, class activities (i.e., practice problems and quizzes), and YouTube videos (where AI experts talk about applications, ethical issues, etc.) were


In this assignment, which is part of the course project, you are supposed to write a one-page letter to a hypothetical editor of a hypothetical periodical.

1. Come up with the name of your periodical. It can be an existing one, such as Seattle Times or a magazine, or something that you made up. It would be great (and have bonus points) if you choose an existing periodical that is consistent with the subject of your project.
2. Write a one page letter to the editor of the periodical selected in step 1. Your letter, mainly talks about the benefit of using AI for the subject of your project, and can be oriented in one of the following ways:
   - You can imagine that the periodical has published an article about the subject of your project, and you want to comment on that article.
   - You can suggest that the periodical publishes a special issue about the subject of your project, with articles about the use of AI to address the corresponding issues.

Figure 4: Letter to the editor assignment.

mentioned as positive points of the course. Students were also interested in AI applications and ethical issues.

In the final exam, where learning outcomes L1, L2, and L3 were covered, 71.3% of the students had a grade above or equal to 80 (out of 100). In the midterm exam, where learning outcomes L1, L2, and L4 were covered, 71.4% of the students had a grade above or equal to 80. Students had the highest performance on learning outcome L4 (i.e., AI applications), which was expected as there is no need to work with any algorithms.

6 Conclusions

In this paper, we introduced a new curriculum to teach AI to non-STEM students. The course covers various AI models and concepts, describes different AI applications, and talks about ethical issues of AI. The course is designed for the students with limited dependency on math and no requirement for computer programming. Ultimately, the goal of this course is not to prepare students to design a complex AI system. The goal is that students understand what an AI system is, be familiar with major components of an AI system, have a clear vision about the underlying algorithms in an AI system, be familiar with the applications of AI, and understand the global issues connected with AI. As demonstrated via some examples, these goals can be achieved with no mathematics, science, and computer programming background.

Acknowledgement

We would like to thank the CS department and the College of Science and Engineering curriculum committees as well as the Core curriculum committee of Seattle University for their feedback during the course design. Furthermore, we would like to thank Dr. Roshanak Roshandel for her valuable comments on this course.

References

[1] Bita Akram et al. “Towards an AI-Infused Interdisciplinary Curriculum for Middle-Grade Classrooms”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.11 (June 2022), pp. 12681–12688.

[2] Safinah Ali, Daniella DiPaola, and Cynthia Breazeal. “What are GANs?: Introducing Generative Adversarial Networks to Middle School Students”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.17 (May 2021), pp. 15472–15479.

[3] Edmond Awad et al. *Moral Machine*. `https://www.moralmachine.net/`. 2016.

[4] Saptarashmi Bandyopadhyay et al. “Interactive Visualizations of Word Embeddings for K-12 Students”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.11 (June 2022), pp. 12713–12720.

[5] Javier Garcia-Algarra. “Introductory Machine Learning for non STEM students”. In: *Proceedings of the First Teaching Machine Learning and Artificial Intelligence Workshop*. Ed. by Bernd Bischl et al. Vol. 141. Proceedings of Machine Learning Research. PMLR, 14 Sep 2021.

[6] Seonghun Kim et al. “Why and What to Teach: AI Curriculum for Elementary School”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.17 (May 2021), pp. 15569–15576.

[7] Phoebe Lin et al. “Zhorai: Designing a Conversational Agent for Children to Explore Machine Learning Concepts”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.09 (Apr. 2020), pp. 13381–13388.

[8] Duri Long and Brian Magerko. “What is AI Literacy? Competencies and Design Considerations”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20. ACM, 2020.

[9] Narges Norouzi, Snigdha Chaturvedi, and Matthew Rutledge. “Lessons Learned from Teaching Machine Learning and Natural Language Processing to High School Students”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.09 (Apr. 2020), pp. 13397–13403.

[10] Organisation for Economic Co-operation and Development. *Artificial Intelligence in Society*. Tech. rep. Organisation for Economic Co-operation and Development, 2019.

[11] PBS Frontline. *In the Age of AI*. `https://www.pbs.org/wgbh/frontline/documentary/in-the-age-of-ai/`. 2019.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, 2010.

[13] Seattle University. *Seattle University Core Curriculum*. `https://www.seattleu.edu/core/`. 2023.

[14] Thomas Way et al. “Machine Learning Modules for All Disciplines”. In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’17. ACM, 2017.

Code Reading: How Students and Professionals Differ*

Matthew Woerner, David Socha, Mark Kochanski
Computing & Software Systems
School of Science, Technology, Engineering & Math
University of Washington Bothell
Bothell, WA 98011
{woerner,socha,markk}@uw.edu

Abstract

This paper reports on a series of semi-structured interviews and code reading exercises conducted with 10 undergraduate students and 11 professional software engineers in order to better understand how students and professionals differ in how they come to understand a codebase that is novel to them. The goal was to uncover distinctions to help us design teaching activities to help students know how to read source code whose call stack spans multiple methods and files. Students had a more difficult time correctly reading source code than professionals and tended to lack a clear process for tackling code reading exercises. Professionals tended to tackle the code reading exercises more methodically, checked their assumptions, and avoided spending time reading code irrelevant to the exercise presented to them.

1 Introduction

This paper reports on results of a qualitative study exploring the following research questions: How do undergraduate students and professional software developers differ in how they come to understand a codebase that is novel

*Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

to them? This question came to our attention while teaching a new course, Software Engineering Studio, designed to reduce the time and effort students take to become effective in their first software engineering internship or job. The course provides an on-campus internship-like experience for students to learn key aspects of working in teams in an organization evolving more complex codebases than they have encountered before. We taught this course to 90 undergraduate students for seven consecutive quarters from autumn 2021 through spring 2023.

In this course, students work in teams as they progress through a sequence of four onboarding badges that progressively a) increase in size and complexity, b) provide fewer instructions, c) move from individual to team-based work, and d) require students and teams to take on more ownership of the work. After doing the core work of a badge, each student creates a portfolio demonstrating their badge work, and then schedules a 30-minute "badge challenge" Zoom meeting with one or both instructors. The badge challenge is an interactive semi-structured interview [1] in which the instructors' notes from their review of the portfolio form starting points for a conversation with the student about their work, processes, mindsets, assumptions, knowledge, and so on. Each conversation is unique as instructors meet the students where they are at, probing to understand precisely what they know, their assumptions, and their current habits—all in the service of uncovering the particular guidance or information suitable for that student. To help stay grounded in facts, students share their screens showing the exact code, tests, documents, or other work they had done. These badge challenges sometimes lead into unexpected territory and sometimes reveal dynamics about our students and curriculum that were opaque to us instructors despite our decades of teaching in our department. Over the course of several hundred challenges, some particularly interesting issues about student preparation have emerged and been the seeds for further research, such as this paper.

This paper emerged from the second badge students do, whose core work required students to modify a C# codebase of five files containing 221 lines of production code, two files of test code, and one JavaScript Object Notation (JSON) data file. When asked during the badge challenge to explain what a particular set of lines of code did, only a handful of the 90 students followed the call stack to see and comprehend what the lines of source code in the called methods actually did. The rest simply made assumptions about what the called code did, despite our repeated questions about their assumptions. They only navigated to the called code when we asked them to, and almost all needed us to tell them how to do that. Looking at the actual lines of code implementing a called method appeared to be a novel idea to all but a handful of the students. We became curious about how to help our students learn how to read and understand code, and decided to see how professionals would perform in this same exercise.

The remainder of this paper is organized as follows. Section 2 briefly reviews some of the related literature. Section 3 describes our methodology for gathering and analyzing data from 21 research participants. Section 4 presents and analyzes the results. Section 5 summarizes our findings and mentions future work.

2 Background Literature

Reading and understanding each statement of software code that one has not written is an essential skill for developers. Studies indicate this code reading can take anywhere from 35-70 percent of a developer's work time. One study of 78 professionals across 7 projects over 3,148 working hours found that "on average program comprehension takes up 58 percent of developers' time" [9]. Despite code reading's importance, there appears to be a relative lack of resources dedicated to helping growing developers practice their code reading skills [11]. This lack of formal education often places new graduates into a situation where they must quickly try to adapt to the code reading abilities of their colleagues in the high-pressure environment of a workplace. A large-scale study at Microsoft found that there is a gap in tool support for reading code, which puts more pressure on the engineer [8].

Previous studies of code reading revealed that professionals are more proficient than students when it comes to identifying and retaining relevant information needed to solve code reading problems [4, 6]. Another study used functional magnetic resonance imaging (fMRI) to show that the size of the codebase being examined by a programmer increased cognitive load, perhaps due to the increased number of identifiers and symbols needed to be tracked for comprehension [5]. These studies suggest that professionals can process code more efficiently than students and at a lower cognitive load [6]. Some hypothesize that more experienced programmers can focus on relevant information needed to read the code in front of them in order to accomplish these tasks with a lower level of cognitive load [6, 5]. Other studies suggest techniques like sketching (the act of creating a visualization of a programming state) are effective at improving novice programmer's ability to complete code reading problems by helping to manage cognitive load [2].

3 Methodology

Our study used semi-structured interviews and code reading exercises to gather data from 10 undergraduate students and 11 professional software engineers (see Table 1) during spring of 2023. The first author did all interviews, using two weekly meetings with the other authors to discuss and adapt the research. Student participants were recruited from the University of Maryland: College Park and the University of Washington through social media outreach and the interviewer's personal networks. Nine of the ten students had prior software engineering internship experience, and most students were in their third year of their undergraduate program. The professionals were recruited via the interviewer's personal network, and came from a variety of large software companies such as Amazon and Microsoft. All but one professional (P2) had been working professionally as software engineers for at least 3 years, and all but two (P3, P7) had a formal computer science education. All interviews and code reading exercises were conducted remotely via Zoom, and video recorded for later analysis. Our university's Human Subjects Division determined that this research qualified for exempt status, and thus did not need to obtain Institutional Review Board (IRB) approval.

Job Title[1]	S	S	S	S	SS	SS	SS	S2	SS						S2	SS					
Professional Experience[2]	3	4	3	1	24	8	15	6	6	1/2	1/2	1/2	0	1/2	8	20	1/4	1/4	1/4	3/4	1/2
University Attending[3]										M	M	M	M	M			M	M	M	UW	M
Year in University										3	3	3	2	3			2	2	3	4	4
CS/CE degree[4]	G	G	G	G		G	G	G		E	E	E	E	E	G	G	E	E	E	E	E
CS internship	✓	✓	✓	✓			✓	✓		✓	✓	✓		✓	✓		✓	✓	✓	✓	✓
Identifier			P1	P2	P3	P4	P5	P6	P7	S1	S2	S3	S4	S5	P8	P9	S6	S7	S8	S9	S10
Exercise Given			E1	E1	E1	E1	E1	E1	E1	E1	E1	E1	E1	E1	E2	E2	E2	E2	E2	E2	E2

Figure 1: Interviewee demographics. [1]Software Engineer (S); Software Engineer II (S2); Senior Software Engineer (SS). [2]Years. [3]University of Maryland (M); University of Washington (UW). [4]Has graduated from (G) or is enrolled in (E) a computer science or computer engineering degree program.

3.1 Semi-Structured Interviews

The semi-structured interview [1] consisted of 14 open-ended questions such as "Can you tell me about how you come to understand code you have never seen before?" and "What do you find makes it easier or more difficult to onboard to a new codebase?" These questions were designed to try and understand how the participant thought about and did code reading, what made the process easier or more challenging, how participants viewed their own approaches to code reading in different contexts (e.g. debugging an issue, onboarding to a new codebase), how those approaches changed over time, and what caused those changes to occur. As usual for semi-structured interviews, the 14 questions served as a set of example questions to guide the interview. The particular questions asked was decided at the time of the interview, and questions were re-arranged, re-worded, or not used depending on the direction of the conversation and thus varied for each interview.

3.2 Code Reading Exercises

After doing two interviews, we added a code reading exercise (E1) to the end of each interview. The exercise used the code repository from the Software Engineering Studio course. Participants were tasked with tracing through a particular unit test to identify what was causing the test to fail, and were allowed to search the internet during their time, and to ask clarifying questions. Participants could solve this problem by following the execution of the unit test's code from the test data file to the class that the test was calling. From there participants could examine the class' initialization in the class constructor, as well as the method within it that was subsequently called, to see that the code read the data file into a global variable of a list of quotes and then searched the list for the first entry that matched the GUID (globally unique ID) provided in the test. All a participant would then need to do in order to find the issue would be to examine the data file itself and see that the information in the data file tied to the GUID used in the test did not match the result that the test was expecting, causing the unit test to fail. If a participant following the path of code calls they would navigate down two separate call stacks before encountering the problem.

To help us understand what they were thinking, participants were asked to talk aloud through their thought process. When participants stopped talking, they were prompted with questions about their thought process. If a participant was not making progress after a significant amount of time, the interviewer provided guiding hints to steer them towards the answer. Participants that appeared unable to solve the exercise and thus were given specific guidance towards the answer had their times logged, but were marked as not completing the exercise unassisted. The interviewer also used the interviewee's cursor movement as a supplementary factor to give hints about the participant's thought process. The interviewer took notes on all participant spoken thoughts and behaviors deemed relevant to the exercise. Most notes ended up focusing on similarities to and deviations from a direct route to the faulty code.

After the 14th interview, we redesigned the code reading exercise to see if the differences between students and professionals revealed from E1 became more extreme with a more difficult exercise. Exercise 2 (E2) added another layer of complexity by asking participants to examine a series of nearly identical passing tests and to determine how to make exactly one test fail by adding lines to a file in the codebase. The starting point for E2 was an additional layer up the call stack from the starting point for E1, and following the code calls would require navigating three separate call stacks, which also added to the difficulty of the exercise. A correct reading of the code would have participants tracing down to the area where the test data file is read and realizing that the code searches for the first element in the data file with a matching GUID. The participant could suggest adding to the start of the data file a new element with the same GUID as the test they wanted to fail but with different data; this would cause data to be populated from the new entry, causing the test to fail while all others continue to pass.

4 Insights From the Interviews

The semi-structured interviews revealed several differences and some similarities between the students and professionals. Both spoke of the importance of documentation and how it helps improve their ability to read code. However, students and professionals tended to place greater emphasis on different types of documentation as being the most helpful. Students tended to emphasize the importance of "in-method comments" which explain what the next few lines in a method are designed to do, similar to pseudocode. Professionals more often used higher level documentation such as official library/package documentation, one-pager feature design documents, or Swagger pages which were used to provide context for their reading of lower level code. Professionals also emphasized the importance of method headers and good method and variable naming as the key low-level components to their understanding of code.

Students and professionals described different approaches to debugging an issue in unfamiliar code. Professionals tended to discuss the importance of narrowing down the bug's location by examining logs and metrics, and would only step through the code if the bug cannot be found with logs alone. Students highlighted a similar high-level approach of narrowing down the search area, but tended to rely on inserting print statements or making minor code changes to see their effect on the output, rather than examining error logs. Roughly half of the students interviewed made no mention of stepping through code with a debugger, whereas all professionals did so.

5 Results From Code Reading Exercises

Figure 2 charts (a)-(c) show the incidences of three particularly interesting anti-patterns that we noticed during the code reading exercises: 1) examining irrelevant files, sometimes apparently randomly choosing different source, data, and configuration files; 2) not following the call stack through files; and 3) making uncorrected misinterpretations of the code. The three anti-patterns were identified during the first few interviews, discussed by all authors who also noted that the same anti-patterns were common during student challenge meetings, and then quantified by the interviewer reviewing the interview videos. Chart (d) shows the total number of anti-patterns each participant exhibited. All but one of the 37 anti-patterns instances were from students, and each student exhibited between 2 and 7 instances.

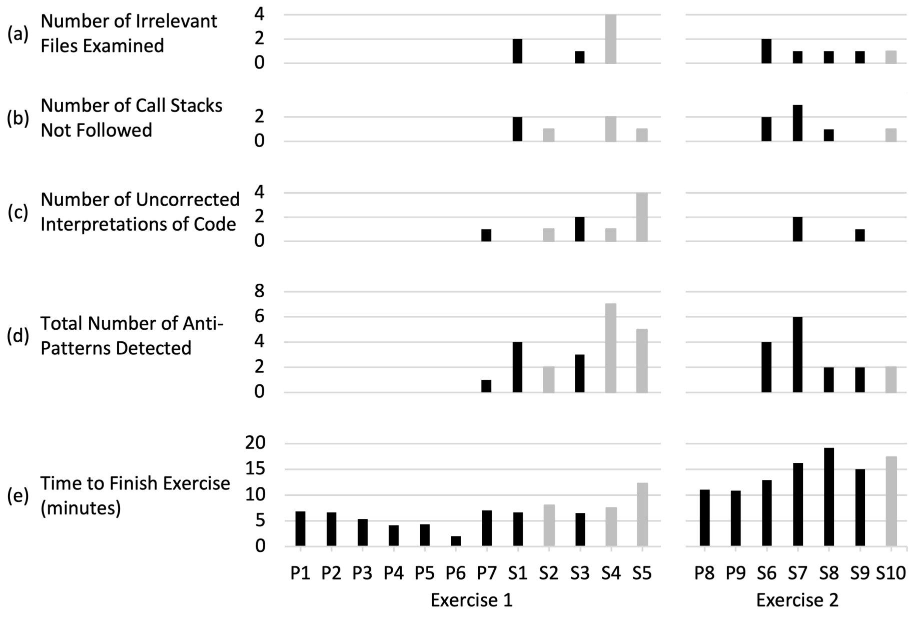


Figure 2: Differences exhibited between students ("S") and professionals ("P") during code reading exercises as compared to the ideal solution. Gray bars denote participants who did not finish.

Figure 2 chart (e) shows the time taken to solve the code reading exercises. For the simpler E1, students on average took longer than the professionals; only two students (S1, S3) took a few seconds less than the slowest professionals (P1, P2). For the more complex E2, all five students took longer than the two professionals. As predicted the more complex E2 took longer to solve.

For E1, participants that appeared more confident in their approach to the exercise tended to speak up about their thought process unprompted, as opposed to participants who showed signs of having difficulty with the exercise. All E1 professionals solved the exercise with minimal deviations from the most direct path to the solution, including four who had no C# experience. Students had varying degrees of difficulty, and all requested assistance. Four out of the five students did not demonstrate a good ability to trace calls through multiple files; many scrolled though irrelevant files outside of the code path looking for a method, or becoming lost and needed assistance finding a method in a different file. This issue could be for a number of reasons, but we hypothesize that the likeliest causes are lack of understanding of how the code was structured in files, and a lack of knowledge of code navigation tools.

Once students found the relevant method the test was calling, four out of the five students assumed that the global variable holding the list of quotes had nothing to do with the error and ignored it until given a suggestion to look deeper into it. One professional also assumed it was irrelevant to the exercise at hand, but self-corrected within 2 minutes without prompts or hints.

Students demonstrated difficulty in finding relevant information within a file they were examining if that information was located in another method or in the constructor. Part of this could be due to the use of a singleton design pattern to instantiate the class, but it still demonstrates a difference and an area where education could be useful.

All five E2 students abandoned the code path to explore files irrelevant to the exercise. E2 students were less likely to misinterpret the code than were E1 students, despite most of the code being largely the same. This could be due to the fact that E2 students more frequently searched the internet for information or asked questions than E1 students, with all but one E2 student seeking additional information on critical library functions they were unfamiliar with, even though the function in question was a part of both exercises. On average, the E2 professionals took 2.1 times longer than E1 professionals, but never got stuck and always kept track of relevant information throughout the process.

Professionals appeared more confident in solving code reading exercises than students and appeared to use a clear methodology. Professionals rarely had to be prompted to speak what they were thinking, while students were far more likely to have long periods of silence (e.g., 30 seconds) during which the interviewer had to prompt them to think aloud. Furthermore, while we have not analyzed this in detail, professionals tended to use key terms indicating that they had a conceptual model of how to read code. Professionals stuck to the flow of code execution and checked any assumptions made for correctness before proceeding. Professionals that were unsure of a language feature or outside function always asked questions or looked up documentation rather than assuming what the thing did. Many of the deviations from this manner of thinking by students appeared to be a result of lack of knowledge, and to students appearing to not use a defined methodology to help guide them through the process.

6 Conclusion and Future Work

This paper reports on the results of a qualitative inquiry into differences between how undergraduate students and professional software developers come to understand a codebase that is novel to them. While our data come from only 21 interviews (10 students, 11 professionals), Figure 2 demonstrates noticeable differences between how students and professionals solved the two code reading exercises. Our hypothesis is that key reasons professionals performed better include that they checked their assumptions about what code did, and understood how to trace code through the execution stack.

Our next step is to design and test a set of teaching activities, sample codebases, and tools to make these qualities salient to students and to give them practice avoiding the anti-patterns shown in Figure 2. Unlike much of the prior work on teaching code reading that mainly focuses on smaller snippets of code [10, 3, 7], we aim to create teaching activities to help students read code in larger codebases like those they will encounter in the workplace. If appropriate, we may include exercises that rely on a form of trace sketching [2] in order to help students understand specific patterns in code.

We believe that reading code is a skill that can be effectively taught to undergraduates by providing them with key simple models of code reading techniques based upon what professionals do, exercises in which to practice those techniques, and a summative assessment exercise to see how well the models and exercises worked to improve students coding abilities. We continue to mine our interview information to inform the design of this and other work which we look forward to reporting on in the future.

References

[1] William C. Adams. “Conducting Semi-Structured Interviews”. en. In: *Handbook of Practical Program Evaluation.* John Wiley & Sons, Ltd, 2015, pp. 492–505.

[2] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. “Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw”. In: *Proceedings of the 2017 ACM Conference on International Computing Education Research.* ICER ’17. New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 164–172.

[3] Cruz Izu and Claudio Mirolo. “Comparing Small Programs for Equivalence: A Code Comprehension Task for Novice Programmers”. en. In: *Proceedings of the 2020 ACM Conference on Innovation and Technology*

in Computer Science Education. Trondheim Norway: ACM, June 2020, pp. 466–472.

[4] Sarah Jessup, Sasha M. Willis, Gene Alarcon, and Michael Lee. "Using Eye-Tracking Data to Compare Differences in Code Comprehension and Code Perceptions between Expert and Novice Programmers". en. In: *Proceedings of the 54th Hawaii International Conference on System Sciences.* 2021.

[5] Norman Peitek, Sven Apel, Chris Parnin, Andre Brechmann, and Janet Siegmund. "Program Comprehension and Code Complexity Metrics: An fMRI Study". en. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* Madrid, ES: IEEE, May 2021, pp. 524–536.

[6] Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. "Measuring neural efficiency of program comprehension". en. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* Paderborn Germany: ACM, Aug. 2017, pp. 140–150.

[7] Leigh Ann Sudol-DeLyser, Mark Stehlik, and Sharon Carver. "Code comprehension problems as learning events". en. In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education.* Haifa Israel: ACM, July 2012, pp. 81–86.

[8] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. "How do software engineers understand code changes? an exploratory study in industry". In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering.* FSE '12. New York, NY, USA: Association for Computing Machinery, Nov. 2012, pp. 1–11.

[9] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. "Measuring Program Comprehension: A Large-Scale Field Study with Professionals". en. In: *IEEE Transactions on Software Engineering* 44.10 (Oct. 2018), pp. 951–976.

[10] Benjamin Xie, Greg L. Nelson, and Amy J. Ko. "An Explicit Strategy to Scaffold Novice Program Tracing". en. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education.* Baltimore Maryland USA: ACM, Feb. 2018, pp. 344–349.

[11] Tammy Xu. *Reading Code Is an Important Skill. Here's Why. | Built In.* en. Nov. 2021.

Generative Pre-Trained Transformer (GPT) Models as a Code Review Feedback Tool in Computer Science Programs[*]

Aaron S. Crandall, Gina Sprint, and Bryan Fischer
Department of Computer Science
Gonzaga University
Spokane, WA 99258
{crandall,sprint,fischerb}@gonzaga.edu

Abstract

Undergraduate computer science and software engineering students benefit significantly from in-depth reviews of their code early and often in their courses. Performing these reviews is time-consuming for teaching assistants and professors to complete, consequently impacting the timeliness and consistency of the provided feedback. When code feedback is not delivered close to the time of authorship, the utility of the review for students is diminished. Prior work with Automatic Static Analysis Tools has shown promise at using artificial intelligence to automate code reviews, with some success integrating them into classroom environments. To leverage new advances in Generative Pre-Trained Transformer (GPT) models, this work reports on an Automatic Review Tool (ART) to provide timely, automatically generated code reviews. ART was evaluated in a second-semester computer science course by integrating ART into the course's Github-based assignment submission system. A cohort of student volunteers (N = 74) read the ART reviews and provided feedback using a survey spanning two of their course assignments. The results of this pilot study show that students perceived ART was successful at detecting defects and offering style-based suggestions, and students were receptive to receiving future automated reviews of their work.

[*]Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

The introduction of Generative Pre-Trained Transformer (GPT) systems has raised serious questions about the role of software developers and code creation processes, both in industry and in the classroom. The greatest focus on GPT's role in software development has been on its use for code synthesis. This tool is able to generate mostly working source code given English language requirements, which can solve many introductory computer science course problems [13]. The field of computer science education needs to explore ways to integrate GPT and other artificial intelligence-based tools into pedagogical practices, just as the software development industry is seeking strategies to leverage GPT-style tools into day-to-day workflows. The modern code review is one such industry practice that lends itself well to automation with GPT. The code review process discovers code defects [12] and helps train less experienced software developers on industry practices [5]. However, since code review is typically done by other programmers, it can be time-consuming to implement at scale. In this paper, we present and evaluate a custom tool that integrates the OpenAI GPT 3.5-turbo model with the code review process to automatically review student code in undergraduate computer science courses. Outsourcing (via machinesourcing) this traditionally human-performed work to a GPT model can introduce several risks and issues, especially when used in an academic setting. This work investigated the following three concerns of using a GPT model for automated code review in a learning environment:

1. How often does the model deliver correct suggestions, incorrect suggestions, and suggestions beyond the student's current knowledge level?
2. What types of code review topics does the model successfully address?
3. How much value do the students perceive from having automated code reviews and do they want the reviews as part of their regular workflow?

To lay the groundwork for our investigation, we designed and implemented a GPT-based software package called Automatic Review Tool (ART) that was integrated into the programming assignment workflow used in a second-semester computer science course. Students voluntarily ran the ART workflow with their code, received an automated code review, and then provided feedback regarding the review. The survey questions were centered around the three concerns above, focusing on suggestion correctness and knowledge scaffolding, topic coverage, and value to students.

2 Related Works

Software engineers and researchers in the field of Automatic Static Analysis Tools (ASATs) investigating code reviews have primarily focused on what makes a ‘good’ code review, how to best teach code review skills in computer science courses, and how to streamline or fully automate the code review process. While the components of a ‘good’ code review are debatable, there are several key parts that are often cited as valuable [12, 14], such as reviewing for style, structure, architecture, execution errors, and testing issues.

Given the value of code review in collaborative software development, there are several recent studies investigating how to teach code reviewing skills in the classroom. Approaches such as using checklists to guide new learners [4], utilizing peer-based code review strategies [7], and showing that using code review in courses improves student outcomes as software engineers [15]. Industry leaders are also continually promoting inclusion of code review training in academic settings [17]. These works show that students are interested and will benefit from exposure to code review tools, strategies, and experience.

Due to the cost of the feedback process, software engineers and researchers have proposed several approaches to automate code reviews in the field of ASATs [16]. To streamline code feedback, semi-automated techniques have included using static analysis tools such as linters and plugins to speed up grading processes [2]. More fully automated [1] tools have shown some success at grading student code, targeting specific areas such as variable naming suggestions [6], training models on prior student work to generate new reviews [3], or using large pre-trained networks [11]. These tools have been incorporated and tested in classroom environments, but are not widely used on a large scale yet [3, 6, 9, 11].

This paper presents a software package, ART, that scales to learning environments with large enrollments and to industry environments with large teams. ART leverages a GPT model [10] built by OpenAI to input student source code and request a code review as output. Our approach contributes to the state of the art in researching and deploying ASATs in educational environments. Specifically, we report on deploying and evaluating a scalable, automated ASAT-style code review package in a pilot study with second-semester computer science students.

3 Methods

To gather feedback about automatically generated code reviews, we conducted a pilot study with computer science students at a primarily undergraduate institution. The pilot study was undertaken in the Spring of 2023.

We conducted this study fully online with Github tools and a survey. Survey respondents were volunteers from two sections (N = 51) of a second-semester course in computer science. This course included an introduction to algorithm analysis, object-oriented programming, linear data structures (linked lists, stacks, and queues), and recursion using the C++ programming language. In addition to these fundamentals, the instructor of these sections also introduced the basics of several software engineering tools, including Git and Github for version control and Google Test for unit testing. In total, there were nine programming assignments throughout the semester. Each programming assignment was submitted via a Github Classroom assignment, which required the students to learn how to make Git commits and push code changes to a Github repository. Beginning with programming assignment #3, the instructor provided starter code with Google Test unit tests for students to run locally to evaluate their code for functional correctness. Furthermore, as part of an auto-grading workflow, these unit tests were automatically executed via a Github Action workflow every time a student pushed their code to Github. Github Actions is a feature of Github that allows DevOps-style software development workflows to run on Github when an event, such as a push or issue creation, occurs in the repository. Students were shown how to navigate on Github to see the details of the auto-grading Github Action workflow.

Towards the end of the course, we augmented the Github Action usage in this course to include the ART workflow that students could voluntarily run as part of an extra credit task on the final two programming assignments, programming assignment #8 (an objected-oriented doubly linked list and queues assignment) and #9 (a recursion assignment). Instructions were provided to students on how to manually run the ART workflow on the Actions tab of their assignment Github repository. The workflow launched a self-hosted Github Action runner that cloned the student's code, executed a Python3 script to make requests to the OpenAI API for reviews of (pre-selected) code files, then displayed the code review as an easy-to-read Markdown summary on the workflow page. A PDF artifact was also generated that students could download and read offline. The workflow process would take about 45 seconds to complete.

To elaborate on our use of OpenAI's API in ART, we utilized the GPT 3.5-turbo model via the official OpenAI Python3 library and web API to generate a code review of students' programming assignment work. The Python3 script pulled the code from the student's repository and added the prompt options shown in Table 1 to query the GPT 3.5-turbo model for the code review. Due to API request prompt/completion token limits for the GPT 3.5-turbo model, one source file was used in a prompt at a time. The prompt was designed to strongly reinforce that the generated text should not include source code to solve the problem for the student, but instead include only prose from a review of the student's code.

To receive extra credit points for the assignment, students were asked to fill out a survey that instructed students to "Please read your generated review carefully and answer the following questions to provide feedback about the re-

Web API field	**Value**
system_content	You are a helpful code reviewing assistant.
starting_prompt	Act as a professional software engineer reviewing code for an undergraduate computer science student. Review this code. Be verbose in your observations. Do not generate new code. Do not give suggested code changes. Do not write revised code. Do not write explanations. Do not give updates to the code. Do not create a revised version with changes. Format everything in Markdown. [Student assigment source code]

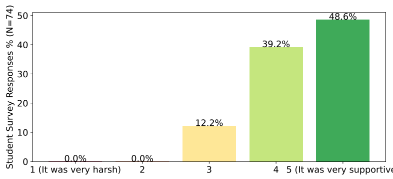
Table 1: OpenAI web API values passed for ART's code reviews.

view of your code. Essentially, you are going to review your code's review!" The survey design was informed by prior work researching types of defects found in code reviews [12], code review quality [8], and code reviews performed in higher education [2, 7, 15], as well as experience reports/blog posts [17]. The survey included several multiple-choice, Likert-style, and free-response questions. Of the 51 students in the course, 40 students (78%) opted to complete the automated code review extra credit task for programming assignment #8, and 34 of these 40 students completed the task for programming assignment #9 as well. In total, our ART pilot study in a second-semester computer science course with student volunteers provided 74 total survey responses.

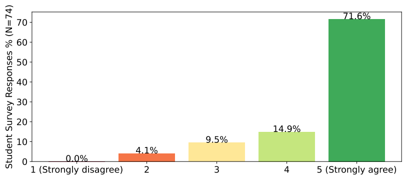
4 Results and Discussion

Overall, the students responded quite positively to the automated code reviews provided by ART. They felt that the 'reviewer' had the proper knowledge to review their code, with 66/74 (89.2%) saying 'yes', and 8/74 (10.8%) saying 'somewhat'. They also felt that the reviewer reviewed the code, and not the student, with an average rating of 4.54/5. This indicator of impersonal writing style is important for professional code reviews and good training for the tone and tenor students should use in their own work performing peer reviews.

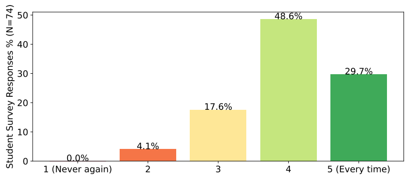
The responses visualized in Figure 1a and Figure 1b show the students agreed and strongly agreed that the reviewer's prose was supportive and professional, which is an important factor to ensure the programmer remains open to constructive criticism during the code review. The students also agreed and strongly agreed that they would want the reviewer to inspect their code again, as shown in Figure 1c, but their desire for a review of this length to be generated on every code version pushed to Github was more mixed, as seen in



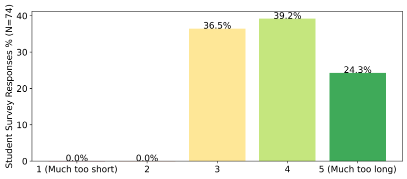
(a) Please rate the review your code received in terms of its tone and use of supportive language.



(b) I felt the reviewer reviewed my code and not me.



(c) Would you want this reviewer to review your code again in the future?

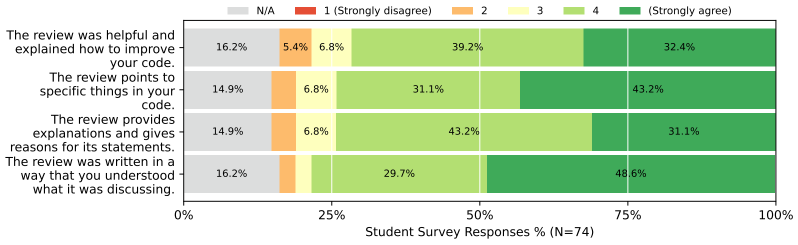


(d) If you received a code review this long on your code with every commit you push to GitHub, how would you find it?

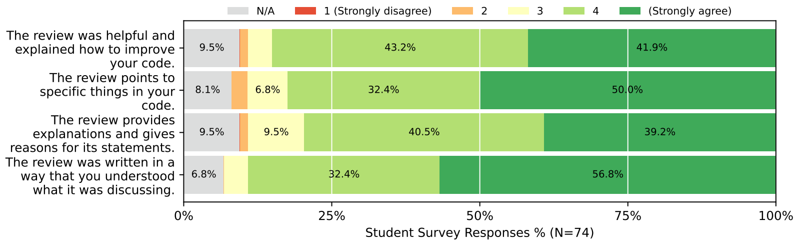
Figure 1: Student responses to Likert questions regarding various aspects of reviews generated by the ART software package.

Figure 1d.

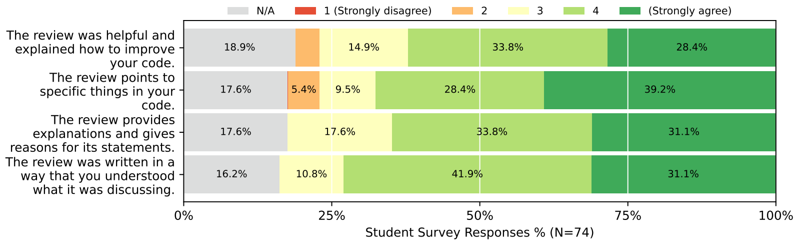
When inspecting the results from the survey questions addressing the specific defects (functional, visual representation, structural, and documentation), the results are positive in most areas. For visual/style defects (Figure 2b) and documentation defects (Figure 2d), students agreed or strongly agreed that the reviewer effectively addressed these areas over 75% of the time. The students were not as confident in the results about functional (Figure 2a) and structural (Figure 2c) defects. For these areas of feedback, the students felt the tool did not address these types of defects (providing a N/A rating), with an overall average of 15.5% and 17.6%, respectively. This was in sharp contrast to visual defects' overall average N/A rating of 8.5% and documentation defects' overall average N/A rating of 2.0%. This is likely due to the model only seeing one source code file at a time, as well as the structure of the student's work being relatively simple in a second-semester course, leaving little room for functional and structural suggestions. It could also be that the GPT model is less effective at high-level code structure analysis than it is at checking conventions and making documentation suggestions.



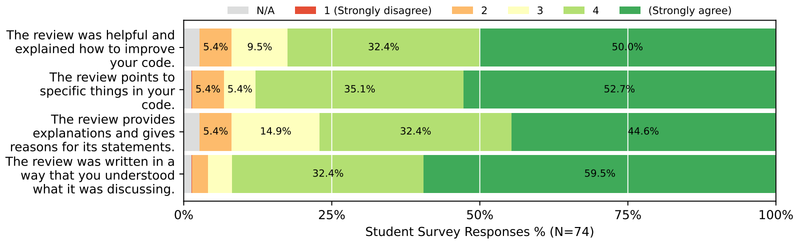
(a) Functional defects.



(b) Visual representation defects.



(c) Structural defects.



(d) Documentation defects.

Figure 2: Student responses to Likert questions regarding code defects.

Assignment	Future ART code reviews	ART reviews every commit
#8	3.94	4.03
#9	4.17	3.65
Rating Change	+0.23	-0.38

Table 2: Likert responses from students participating in both surveys (*boths* group). Scale: 1 - never again/much too short to 5 - every time/much too long.

At a more granular level, there are a few other interesting results from the surveys. One such result is looking at the estimated time it took the students to read and use the code review. Within the programming assignment #8 cohort, there are two subgroups of students. One group chose to participate in both assignment surveys (*boths*), and the other group only participated in the programming assignment #8 survey (*singles*).

The *boths* group and *singles* groups exhibited similar responses to the Likert questions. The one notable difference was how long they took to read and use the review. The *boths* group estimated an average of 8.1 minutes to read their review while the *singles* estimated an average of 12.5 minutes to read the review. Taking an average of 50% longer to read the code review might have been a contributing factor as to why the *singles* chose not to participate in the second survey opportunity, despite an overall positive desire to get future code reviews. We observed another notable outcome when comparing the *boths* group survey results from programming assignment #8 to #9. As shown in Table 2, the students reported an increased desire for code reviews in the future from ART, but less desire for a review of this length on every commit to GitHub.

5 Conclusion

GPT models show great promise in providing natural language processing and generation in the field of Automatic Static Analysis Tools. Integrating these tools into software development workflows and educational learning frameworks will be an ongoing area of research and development. This work piloted the use of a GPT model as an ASAT-style code review package for helping second-semester computer science students improve their own code and for exposing them to a professional-style code review process.

Our ART software package demonstrates an efficient way to give timely feedback during course assignments alongside current teaching assistant/faculty assignment grading processes. Results from a pilot study with ART showed a strong interest from students to have these tools in their course support system. Specifically, our findings indicated the automatically generated reviews were mostly within the learned scaffolding of second-semester computer students and addressed most areas expected from professional code reviews. The students felt the reviews were generally helpful and would want them to be available for future assignments. Overall, the ART reviews were accurate in their suggestions and showed promise as a powerful addition to the pedagogical landscape of software engineering courses.

In the future, we plan to perform additional testing and evaluation of ART. The next iteration of ASAT-style tools for code review should include a more in-depth evaluation of code review accuracy and the ability to give structural suggestions, as well as attempt to normalize the style of the output. These tools should also be updated to handle custom code style guides so they do not provide conflicting suggestions compared to the course guidelines. Finally, additional work should measure the longitudinal impacts of these tools across the four-year curriculum and in the early stages of software development careers.

References

[1] Kirsti M Ala-Mutka. "A survey of automated assessment approaches for programming assignments". In: *Computer Science Education* 15.2 (2005), pp. 83–102. DOI: 10.1080/08993400500150747.

[2] Mary Elaine Califf and Nick Dunne. "Feedback in Context: Using a Code Review Tool for Program Grading". In: *ACM Technical Symposium on Computer Science Education.* Vol. 1. 2022, pp. 92–97. DOI: 10.1145/3478431.3499402.

[3] Robert Chatley and Lawrence Jones. "Diggit: Automated code review via software repository mining". In: *International Conference on Software Analysis, Evolution and Reengineering.* SANER. IEEE. 2018, pp. 567–571. DOI: 10.1109/SANER.2018.8330261.

[4] Chun Yong Chong, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. "Assessing the students' understanding and their mistakes in code review checklists: an experience report of 1,791 code review checklist questions from 394 students". In: *International Conference on Software Engineering: Software Engineering Education and Training).* IEEE. 2021, pp. 20–29. DOI: 10.1109/ICSE-SEET52601.2021.00011.

[5] Felipe Ebert et al. "Confusion in code reviews: Reasons, impacts, and coping strategies". In: *International Conference on Software Analysis, Evolution and Reengineering.* SANER. IEEE. 2019, pp. 49–60. DOI: 10.1109/SANER.2019.8668024.

[6] Elena L Glassman et al. "Foobaz: Variable name feedback for student code at scale". In: *Annual ACM Symposium on User Interface Software & Technology.* 2015, pp. 609–617. DOI: 10.1145/2807442.2807495.

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

[8] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. “Code review quality: How developers see it”. In: *International Conference on Software Engineering.* 2016, pp. 1028–1038. DOI: 10.1145/2884781.2884840.

[9] Harsh Lal and Gaurav Pahwa. “Code review analysis of software system using machine learning techniques”. In: *International Conference on Intelligent Systems and Control.* ISCO. IEEE. 2017, pp. 8–13. DOI: 10.1109/ISCO.2017.7855962.

[10] Chuan Li. *OpenAI’s GPT-3 Language Model: A Technical Overview.* Accessed 2023.06.20. 2020. URL: https://lambdalabs.com/blog/demystifying-gpt-3.

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

[12] Mika V Mäntylä and Casper Lassenius. “What types of defects are really discovered in code reviews?” In: *IEEE Transactions on Software Engineering* 35.3 (2008), pp. 430–448. DOI: 10.1109/TSE.2008.71.

[13] Ben Puryear and Gina Sprint. “Github Copilot in the Classroom: Learning to Code with AI Assistance”. In: *J. Comput. Sci. Coll.* 38.1 (Nov. 2022), pp. 37–47. ISSN: 1937-4771.

[14] Caitlin Sadowski et al. “Modern code review: a case study at google”. In: *Iinternational Conference on Software Engineering: Software Engineering in Practice.* 2018, pp. 181–190. DOI: 10.1145/3183519.3183525.

[15] Xiangyu Song, Seth Copen Goldstein, and Majd Sakr. “Using Peer Code Review as an Educational Tool”. In: *Conference on Innovation and Technology in Computer Science Education.* 2020, pp. 173–179. DOI: 10.1145/3341525.3387370.

[16] Carmine Vassallo et al. “Context is king: The developer perspective on the usage of static analysis tools”. In: *International Conference on Software Analysis, Evolution and Reengineering.* SANER. IEEE. 2018, pp. 38–49. DOI: 10.1109/SANER.2018.8330195.

[17] David Kofoed Wind. *Teaching Code Review to University Students.* Online. Accessed 2023.06.19. Mar. 2020. URL: https://www.eduflow.com/blog/teaching-code-review-to-university-students.

A Course Model for Ethics Education in Computer Science[*]

Ben Tribelhorn and Andrew Nuxoll
The Donald P. Shiley School of Engineering
University of Portland
Portland, OR
{tribelhb,nuxoll}@up.edu

Abstract

This paper presents a course model for teaching ethics to Computer Science students without being linked to a technical topic which helps students think more holistically. The course model proposed allows for students to build on their philosophy of ethics course from the core and to focus on current issues in technology. The assignments and topic generation activities presented can be easily adopted, even into other courses for institutions that don't offer technology focused ethics course. This course assists in aligning the Computer Science major curriculum with one of the five ABET program outcomes and to align with current guidance from ACM. The course was evaluated with a student survey to assess five learning objectives. The initial survey results show that students felt they improved on all five, especially in their ability to identify ethical issues in design decisions.

1 Introduction

Technology and software systems play an ever growing role in our lives. Computer Science (CS) students will inevitably be making design decisions in their careers that impact large swathes of society. With advances in artificial intelligence, such as ChatGPT, self-driving cars, and more, the choices that may face

[*]Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

these students could have large and disparate impacts. These concerns have motivated a variety of educational approaches for helping to prepare today's students for these probable future challenges.

Ethics education and design focused education both appear in Computer Science curricula, given requirements for ABET accredited programs[1] and recommendations from ACM's Curriculum Guidelines[2]. The renewed focus on these areas is apparent in the CS2023 ACM/IEEE-CS/AAAI Computer Science Curricula where the *Society, Ethics and Professionalism* subgroup recommends 17 hours of core hours and an additional 14 if it is a "Knowledge Area" focus for a CS program (as of March 2023)[3]. As these are lecture hours, it is easy to see that this content comes close to being a full course. Given the importance of these topics, we present our course model for a 2-credit (semester) course that allows students to thoroughly engage with this content. We present the results of an IRB-approved post survey to see if the students reported growth in their abilities to analyze and synthesize.

2 Course Model

2.1 Background

The key topics listed in ACM's CS2023 are Social Context, Methods for Ethical Analysis, Professional Ethics, Intellectual Property, Privacy and Civil Liberties, Professional Communication, Sustainability, History, Economies of Computing, Security Policies Laws and Computer Crimes, Equity Diversity and Inclusion. Fiesler et al. show what we really teach in "Tech Ethics" courses[6] and it becomes clear that there is a lot of variety in approaches and foci and the perspectives used are not all all Western as there is global interest in this area[7]. Given the degree of activity in the field right now, it may be a good time to revisit approaches to teaching ethics in CS.

Many institutions teach these topics across courses, as there are often natural places to include discussions of these topics, such as in a Software Engineering course and technical electives. Saltz et al. argue "that ethics content should be integrated into core computer science classes, as a preferable solution over simply having a standalone ethics class[10]." Others use a technical course to focus on these topics, for example Skirpan et al. showcase embedding many of these in a Human Centered Computing class[12] and Shapiro et al. embed deeply in their data-focused Human Computer Interaction course[11].

And the third approach is to have a course focus on the deeper context of ethics education. Raji et al. argue that many current approaches use an "exclusionary pedagogy," where ethics is distilled for computational approaches but fails to connect to epistemological foundations[9]. We favor this argument because we want our students to be prepared for future unknowns and they can only do so if they have worked to develop their own ethical foundation. We also align with some of Stalz's argument as our curricular structure includes two ethics courses, with the second one being CS and technology focused. Our design also allows for immediate adaption to current topics. Borenstein and Howard argue that there needs to be AI ethics education[5], and as current concerns change, so must an ethics focused course.

2.2 University Context

Our institution has a robust core curriculum and all of our students must take at least one traditional, philosophy of ethics course, and tend to take at least one perspectives course that focuses on expression and personal foundations. By the time students are Juniors they usually have completed these courses and are eligible to take our "Computer Science Seminar" course. It is a 2 credit semester long course that is required for the CS major and requires Junior standing. It meets for 2 hours a week for 15 weeks and is typically limited to not more than 20 students per section. Many other courses in the curriculum also cover ethical issues, so this course is able to take a non-technical approach to the topics and focus on design decisions more generally.

2.3 Design

Our course model is a variety of work, to help foster engagement for different types of learners. We begin the course by going over each assignment type and participation expectations. Then on the second day of class, we invite students to generate a list of current issues in technology. Each idea is written on the white boards by the instructor. After about 30-50 issues have been identified, the students get three votes to spend on the "most important" ones. The top ranked issues are then used along with instructor required topics to generate a topics list for the section. These can range from very specific: "Elon Musk," to very broad: "data collection." Each student is then assigned to two in class events based on these topics, so that each event covers a unique topic. In the following we divide the classwork by the primary modality. Most class periods operate as students presenting or debating followed by group discussions moderated by the instructor.

An important part of evaluating opinions on an ethical issue is understanding bias. The growing predominance of propaganda in our national discourse has only strengthened this perception. To this end, one of the early lectures in the course is focused on this topic. Students learn the CRAAP test [4] (Currency, Relevance, Author, Accuracy, Purpose) for evaluating the bias in a given source. Student citations in all their subsequent work are strictly evaluated with the CRAAP test during group discussion and when grading student work. Using this method discourages low effort research by students using an internet search engine (e.g., Google) since such searches often return biased sources.

Presenting to a group. Students present on an assigned topic in pairs (or individually), covering at a minimum: an overview, history including non-technology, current practices, and two topical case studies. The connection to the history of a practice really helps students think about issues as human concern. We prompt them to think of "not just the last 200 years, but all of human history." This connects the students to their prior courses in ethics, philosophy, sociology, and/or history. That foundation helps them contextualize the case studies where they are required to present both sides of the issues without taking a stance. Those stances can then come up during group discussion.

Students also are assigned a debate or mock trial where the exact format is instructor dependent. In the Fall 2022 section, we assigned a mock trial where the students were assigned a variety of roles: direct lawyer, cross lawyer, and witness for both the prosecution and defense. In smaller sections this could be limited to a single lawyer for each side. The lawyers then must agree to a contentious topic to debate one week ahead of the mock trial. The instructor reserves the right to assign or modify a debate claim. Before the trial begins, the whole class rates on a line on the whiteboard where their beliefs are for the argument. Students then make opening statements, ask questions of the witnesses, and then make closing statements. After the debate, the whole class marks a parallel line to see if their beliefs have shifted. In about half of the debates opinions reverse based on increased knowledge, and the other half they become stronger based on the increased knowledge. The phrasing the students pick to argue over is a mix of legally aligned claims and more vague goals. Examples are arguing for common connectors on phones or regulating AI in specific ways. When stuck on a topic, there is usually a real conflict between a pair of companies or a company and government or agency that can be used or modified.

Finally, students complete two other group presentation activities using instructor mandated topics. At the beginning of the course, they are assigned an historical figure in CS that is connected with innovations that have led to the modern field and also had to overcome adversity to succeed (based on their race, gender, sexuality, etc.) They present on these issues and amazing contributions to the field and it helps prime them for candid discussions in the rest of the course. About mid-way through the course the students present their results from a "journalism" project. We assign a general topic such as privacy, and then each team has to pick a specific sub-topic, vetted by the instructor, to research. After their research, they prepare a set of 3-7 interview questions and each team member interviews at least three peers that are **not** computer science majors. This forces them to discuss serious issues with less technically-trained peers and really gets them to push past their comfort. The due dates for the sub-topic, set of questions, and poster (or 3 slides) are scaffolded so that they don't wait to the last second and the instructor can provide feedback if necessary.

An alternative to the Journalism assignment is a Jigsaw style assignment [8]. Students are organized into small teams. Each team is assigned the same broad issue (as per the Journalism assignment) that they must evaluate. They then must prepare a poster that proposes a change to US law (or maintenance of the status quo) with regard to that issue. The conduct their research on this topic, meta-teams are formed based on different perspectives (e.g., contemporary domestic law, international law, united nations recommendations, etc.) Each meta-team consists of one member from each project team. The meta-teams prepare a report which is then brought back to each project team for review. The research and preparation activities span multiple lecture periods. The final posters are presented in a conference style on the last day followed by a group discussion.

Written arguments. Students are assigned a short essay that must argue a contentious issue in technology that was not one they have been assigned to present or debate (but it could be one assigned to someone else). These require the students to cite quality sources and limit their prose so that they are able to make a compelling argument. Upgrades to this assignment could include requiring a visit to the writing center, multiple drafts, or peer-evaluations. We have found each to have trade-offs so instructor preference is typically best. These can be fascinating arguments including the argument that Loot Boxes[1] are not ethical and should be banned, AI generated images are theft, hacking can be ethical, etc. These help the students find an issue they care about and deepen their rationale for their position. In more than one case, the instructor has learned that writing the essay actually changed the mind of the student based on their research!

The second area for writing is in an open everything[2] take home final exam, that encourages students to think about more subtle issues and a variety of issues that weren't covered as deeply in the course. These can include questions on copyright, real world examples of questionable behavior, and more contrived situations. These help solidify that the students have, hopefully, strengthened their ability to make judgements and defend them.

Personal development. The students must individually work on their professional development by completing two different tasks towards that goal.

[1]Paying real money for random goodies in video games, a process that has similarities to gambling.

[2]Open-note, open-book, open-internet, open-classmate, and open-mind.

This can include reading a CS or technology focused book, applying to jobs or internships, community service, a coding project, attend a conference or code-a-thon, etc. The deliverable can be a written reflection or a discussion one-on-one with the instructor. These get students to engage with the field outside of the purely academic context. By following what they find important on this assignment, they are working to shape and understand their values.

The other area that contributes individually is the "participation" which includes talking in group discussions, assigned readings (and the reading quizzes), peer-evaluations on the essays.

Overall these three areas help get the students engaged in a wide variety of topics and a number of methods for communicating about them. This design ensures that different learning styles are available across the course. The group discussions are a key component to getting the students to think more deeply about current issues and their own perspectives. The instructor moderation often includes asking questions if discussion doesn't naturally flow. In larger sections and online during COVID, smaller group discussions were effective to get the students to talk about their own viewpoints.

We believe that an important part of being an ethical individual is speaking up. This can be particularly difficult for some students who struggle with public speaking and would rather passively attend a discussion. To encourage the practice, the instructor overtly tracks the total number of discussions that each student participates in and this maps directly to a quantitative grade. Shy students are encouraged to review the discussion topic beforehand and prepare remarks. Thus, they can select topics they feel best speaking about and avoid the anxiety associated with impromptu speaking.

3 Results

The students were given the option to respond to a survey during class time when the instructor left the room. All students present that day responded, but only 66.7% of students were present on that day. The survey consisted of a question to verify the student was a CS major, and then five paired questions. Each question follows the format "To what extent has your ability to **X** increased during this class?" Followed by "Describe a specific example of how your ability to **X** increased during this class." Where the **X** is the learning objective listed in Table 1. The final question was slightly different as it asked "To what extent have your created or modified your personal ethical code of conduct during this class" and the same change in the associated describe question. The extent question used a Likert scale with responses: 1 Not At All, 2 Very Little, 3 To a Small Extent, 4 To a Moderate Extent, 5 To a Great Extent. The numerical results are listed in Table 1.

Table 1: Summary of survey results for each learning objective in the course, n=14 students.

Learning Objective	**Mean** (n=14)	**Standard Deviation**
Evaluate the impact of design decisions on users of technology	3.50	0.94
Self-reflect on your own ideas	3.29	0.73
Identify ethical issues in design decisions	3.86	1.03
Evaluate design decisions in economic terms	2.57	0.85
Create or modify your personal ethical code of conduct	2.64	1.15

Qualitatively, the students mentioned various specific current events, or more general scenarios of being more aware of topics/issues when reading or seeing something. Almost all of the students were able to describe at least three examples from the five questions, and a majority had an example for all five. This supports the quantitative result that students self-rated a learning gain of small to moderate for design decisions and self-reflection. For the more difficult topics economics terms (a tradeoff to ethical choices in many scenarios) and creating a personal code of conduct, the reported gains were smaller. The qualitative responses mirrored this, as some did not see the economic connection, and at least one student reported they would still take any job if it paid.

4 Discussion

We described a course model for a Computer Science course focused on engaging students with ethical issues in technology and their role in those issues with a focus on design decisions. Our approach used a variety of educational techniques to develop the learning objectives.

The survey results show that the students were able to see how their actions as technologists might have ethical concerns and think about these concerns in terms of design decisions. Having this course follow required foundational courses helps the students apply knowledge learned there to more specific topics in the field they are presumably interested in. The highest rated response ties into the ABET Program Outcome *Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.*. This illustrates that this course design is contributing to an important learning objective. These results also support delivering an ethics focused course instead of incorporating all of these topics into smaller units across the curriculum.

Given the broad scope of the course these results are promising and suggest some areas for continued improvement to encourage students to take the time to focus on their own personal ethical code of conduct. We plan on developing an assignment to get students to write this out and apply it to some scenarios to deepen that learning objective.

Acknowledgements

Thanks to the Kern Entrepreneurial Engineering Network (KEEN) and the Donald P. Shiley School of Engineering for supporting this work and course development.

Please contact the authors for homework assignment write ups and rubrics which will be happily shared.

References

[1] ABET. *Criteria for Accrediting Computing Programs, 2023-2024.* https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2023-2024/. URL: `https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2023-2024/`.

[2] ACM. *Association of Computing Machinery Curriculum Guidelines.* https://www.acm.org/education/curricula-recommendations. URL: `https://www.acm.org/education/curricula-recommendations`.

[3] ACM, IEEE-CS, and AAAI. *CS2023: ACM/IEEE-CS/AAAI Computer Science Curricula.* https://csed.acm.org. URL: `https://csed.acm.org`.

[4] Sarah Blakeslee. "The CRAAP test". In: *Loex Quarterly* 31.3 (2004), p. 4.

[5] Jason Borenstein and Ayanna Howard. "Emerging challenges in AI and the need for AI ethics education". In: *AI and Ethics* 1 (2021), pp. 61–65.

[6] Casey Fiesler, Natalie Garrett, and Nathan Beard. "What do we teach when we teach tech ethics? A syllabi analysis". In: *Proceedings of the 51st ACM technical symposium on computer science education.* 2020, pp. 289–295.

[7] Janet Hughes et al. "Global and local agendas of computing ethics education". In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.* 2020, pp. 239–245.

[8] David V Perkins and Michael J Tagler. "Jigsaw classroom". In: *Promoting student engagement* 1 (2011), pp. 195–197.

[9] Inioluwa Deborah Raji, Morgan Klaus Scheuerman, and Razvan Amironesei. "You can't sit with us: exclusionary pedagogy in AI ethics education". In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency.* 2021, pp. 515–525.

[10] Jeffrey S Saltz, Neil I Dewar, and Robert Heckman. "Key concepts for a data science ethics curriculum". In: *Proceedings of the 49th ACM technical symposium on computer science education.* 2018, pp. 952–957.

[11] Ben Rydal Shapiro et al. "Re-Shape: A method to teach data ethics for data science education". In: *Proceedings of the 2020 CHI conference on human factors in computing systems.* 2020, pp. 1–13.

[12] Michael Skirpan et al. "Ethics education in context: A case study of novel ethics activities for the CS classroom". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education.* 2018, pp. 940–945.

Interpreting Machine Learning-Based Intrusion Detection in IoT using Explainable AI: A Case Study with Explainable Boosting Machine[*]

Yizhou Xu and Parteek Kumar
Department of Computer Science
Whitman College
Walla Walla, WA 99362
{xuy2, kumarp}@whitman.edu

Abstract

In this article, we utilize an explainable AI approach, the Explainable Boosting Machine (EBM), to perform feature analysis on an extensive Internet of Things (IoT) dataset collected from real-world devices. After feature selection and data processing, our training dataset includes 2.88 million traffic data instances, categorized into six classes (DDoS, DoS, Mirai, Recon, Spoofing, and Benign). The EBM trained on this dataset achieved a impressive accuracy rate of 99.4% and an F1 score of 92.8%. Using the resultant model, we interpreted its predictions based on feature importance. The identified feature importance aligned well with established cybersecurity principles, indicating the model's potential. However, our analysis revealed that the machine learning model's predictions were strongly tied to the specific characteristics of the training IoT dataset, thereby raising concerns about the model's reliability when applied to real-world attack detection. Future research could explore the use of more diverse and balanced datasets or the applicability of the machine learning model in different IoT contexts, aiming to enhance the model's generalizability and practical relevance.

[*]Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Machine learning is progressively becoming a fundamental component of intrusion detection systems for the Internet of Things (IoT). Although numerous studies have shown promising results using machine learning, a common limitation is the lack of depth in explaining their methodologies, posing concerns about these methods' real-world applicability. In this article, we harness the power of explainable AI, specifically the Explainable Boosting Machine (EBM), to demystify the predictions made by our machine learning model. The EBM was selected for its robust performance, superior interpretability, and effective visualization capabilities. EBM, which combines the simplicity of linear or decision tree models with the precision of complex models like gradient boosting or random forests, functions as a "glass box" model, allowing both its predictions and its decision-making processes to be interpretable by humans.

We utilized a comprehensive IoT dataset as our training data[7]. Upon training the EBM on this dataset, we achieved an impressive accuracy rate of 99.4% and an F1 score of 92.8%. The feature importance produced by the EBM facilitated the interpretation of the model's predictions.

The major contributions of our work are:

- The decision-making of the machine learning model, trained on the IoT dataset, was interpreted using Explainable AI techniques (EBM).
- We cross-referenced the decisions made by our machine learning model to verify their alignment with fundamental cybersecurity principles.
- After thorough validation, we underscored the limitations of the machine learning model, particularly when trained on a specific dataset.

2 Related Work

The detection of malicious activity through internet traffic data remains a compelling avenue in cybersecurity research, with numerous explorations spanning several years. The evolving complexity of IoT systems has led to an increased adoption of machine learning techniques in the creation of Intrusion Detection Systems. The IoTIDs (2020) study assessed the efficacy of diverse traditional machine learning models, such as Linear Regression (LR), K-Nearest Neighbors (KNN), Decision Tree (DT), Random forest (RF), Support vector machine (SVM), and Naive Bayes (NB), using IoT datasets[4]. In another work[1], the researchers developed an advanced deep neural network that outperformed Restricted Boltzmann machine (RBM), Sparse Autoencoder (SAE), and Stacked Denoising Autoencoder (SDAE) frameworks on an IoT dataset.

Despite extensive research into acquiring comprehensive data from IoT devices and the development of robust machine learning models, a thorough analysis of the trained models is frequently absent. For instance, the authors of the Edge-IIoTset (2021) ranked the importance of features for each class without providing any explanatory commentary[3]. The researchers behind WUSTL-IIOT (2021) determined feature importance based on their impact on the model's accuracy rate[12]. Yet, this traditional method's interpretability remains somewhat restricted. In another study[6], the authors concentrated on using feature importance to eliminate redundant features rather than interpreting the model.

3 Methodology

3.1 Explainable Boosting Machine

To comprehend the underpinnings of EBM, we first inspect the standard mathematical form of Generalized Additive Models (GAM)[5]:

$$g(E[y]) = \beta_0 + \sum f_j(x_j) \tag{1}$$

EBM can be viewed as an extension of the GAM, where the prediction is obtained by summing up individual feature functions, f_j. Feature functions encapsulate each feature's contribution to the overall prediction. In contrast to models like logistic regression, which presumes linear relationships, EBM capture non-linear associations, offering superior performance with complex datasets. Furthermore, by plotting the feature functions, f_j, we can identify and visualize the importance of each feature involved in the prediction[5].

In the mathematical formulation below[5], the algorithm for EBM is further improved to detect the pairwise interaction between features (i, j):

$$g(E[y]) = \beta_0 + \sum f_j(x_j) + \sum f_{i,j}(x_i, x_j) \tag{2}$$

3.2 Procedure and Evaluation

This study employs a comprehensive IoT dataset, obtained from real-world devices within a complex network topology[7]. Following feature selection and data processing, our training dataset comprises 2.88 million traffic data entries, spanning six classes: DDoS, DoS, Mirai, Recon, Spoofing, and Benign traffic. We used the utils class from Scikit-learn[8] to calculate the balanced class weight for each class, incorporating these weights into the EBM's parameters.

The processed dataset was then classified by the EBM. Table 1 presents the performance metrics of the resulting EBM model. The model showcased exemplary performance, achieving an impressive accuracy rate of 99.4% and a commendable F1 score of 92.8%. The recall and precision rates generally

Table 1: Results of multi-class classification using EBM

Metric	Explainable Boosting Machine
Accuracy	0.9940481456770219
Recall	0.9367583492627487
Precision	0.9204079536958586
F1-score	0.9276943954810214

maintained equilibrium, attributable to our effective data balance methods. These scores highlight superior predictive capabilities and overall efficiency. A thorough analysis of feature importance based on the EBM's outputs will be presented in the subsequent sections.

4 Results and Discussion

EBM evaluates feature importance through a raw score. In a classification context, the raw score serves as a measure of the evidence supporting a particular class. A high raw score implies strong evidence in favor of that class, while a lower score may indicate weaker support. To comprehend the nuances of feature importance, we must delve into the plots associated with each feature, taking advantage of EBM's data visualization tools. In Figure 1, the vertical axis denotes the raw score, while the horizontal axis represents the feature's value. The description of each curve's color can be found below Figure 1. We will assess features that significantly influence the model's prediction.

4.1 Inter-Arrival Times (IATs)

A crucial feature in the dataset, IATs, measures the time difference between two packets. Our testing designates this as a significant feature, demonstrating that its removal could reduce the machine learning model's accuracy by 15%. The first plot on the left in Figure 1 illustrates the feature importance of IATs. Data predominantly cluster around three intervals: 0s to 0.04s, 0.0829s to 0.0838s, and 0.1664s to 0.1669s. The first interval, from 0s to 0.04s, can be analyzed without the need to zoom in on the plot. The DoS attack class scores the highest, with scores fluctuating between 2.5 and 4, followed by Benign, with scores approximately 1.4. DDoS and Mirai are the least likely classes, with scores ranging from -0.8 to -2.2. Feature importance in the 0s to 0.04s interval is relatively insignificant for the remaining attacks. The second and third plots on the left in Figure 1 offer a detailed view of the IATs feature within the second and third intervals. In the 0.0829s to 0.0838s interval, we

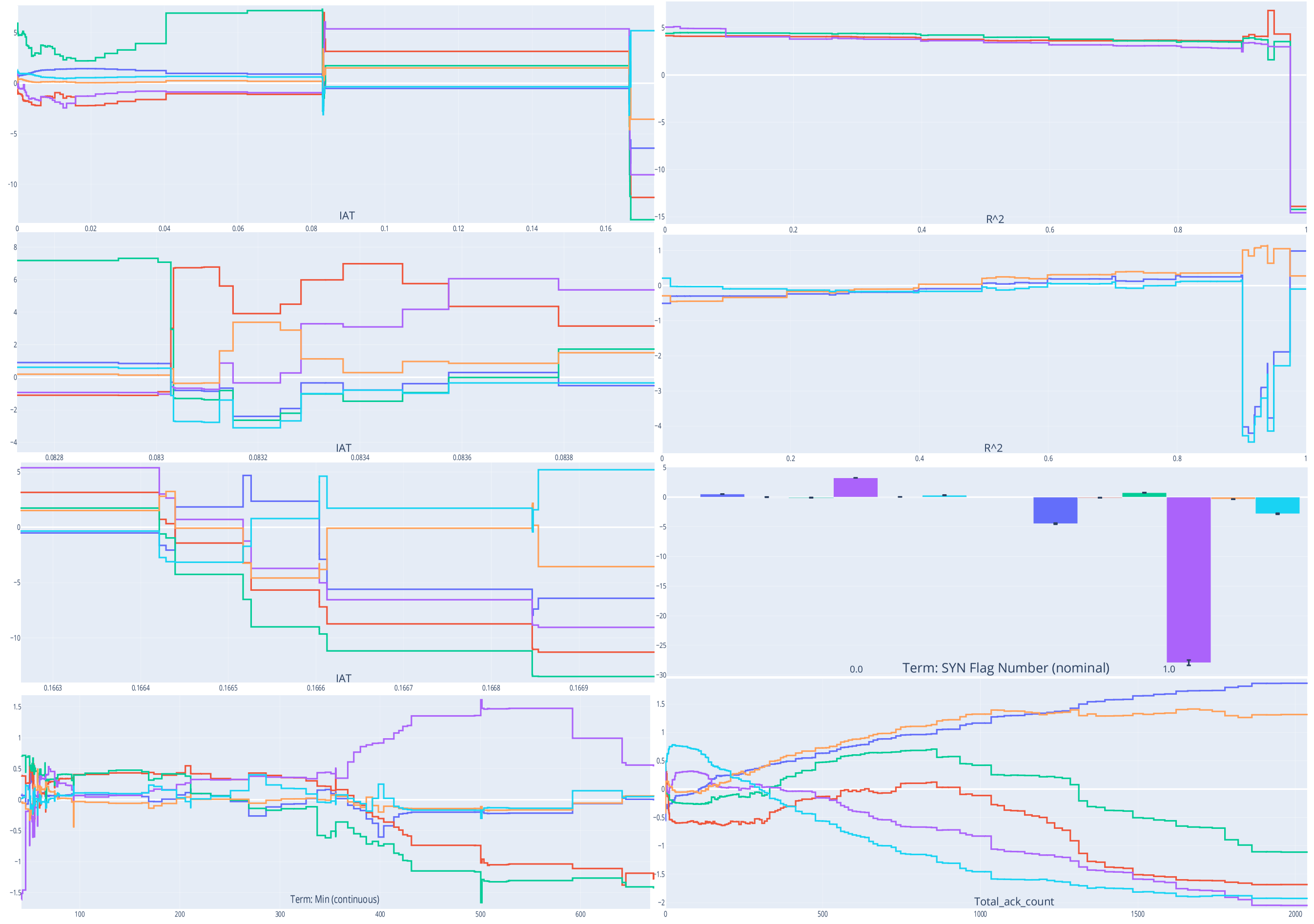


Figure 1: Representations are as follows - DDoS in Red, DoS in Green, Mirai in Purple, Recon in Yellow, Spoofing in Sky Blue, and Benign in Blue.

observe a rising trend in scores for Mirai attacks and consistently high scores for DDoS. Due to the negative feature importance, the machine learning model is less likely to predict DoS attacks within this interval. Lastly, the third plot on the left in Figure 1 shows a declining trend for the scores of DDoS, DoS, and Mirai attacks within the 0.1664s to 0.1669s interval.

These findings can be corroborated with fundamental cybersecurity principles. During DDoS or DoS attacks, attackers aim to overload a network with internet traffic, typically involving sending a large volume of packets in a short span of time. Hence, the IATs during an attack would be exceedingly low, often nearing zero. Mirai attacks involve remote control and device utilization in DDoS attacks, implying similar traffic characteristics to DDoS[2]. Given that these attacks are characterized by shorter IATs, it is improbable for them to reach a duration of 0.16s. This explains why the DoS class has the highest score in the lowest interval (0s to 0.04s) and why there are strong positive scores for DDoS and Mirai attacks from 0.0829s to 0.0838s.

Furthermore, in the second plot on the left in Figure 1, both Benign and Spoofing exhibit a similar pattern of negative feature importance in the 0.0829s to 0.0838 interval. Conversely, strong positive feature importance for Recon suggests a high likelihood of the model predicting this class. In a Recon attack, the attacker might generate noticeable patterns in the network's IATs during scanning. Hence, it is expected to observe high feature importance associated with certain intervals of IATs. In general, IATs analysis can yield valuable insights into understanding the decision-making process of the model.

4.2 The Coefficient of Determination (R^2)

The first and second plots on the right in Figure 1 illustrate the feature importance of R^2. This feature evaluates the coefficient of determination between the lengths of incoming and outgoing packets within a flow. It assesses the correlation between these packet lengths and can yield a value ranging between 0 and 1. During standard network operations, the sizes of incoming and outgoing packets often display a certain correlation. This correlation may, however, change significantly during an internet attack.

The first plot on the right in Figure 1 shows that DDoS, DoS, and Mirai attacks exhibit similar correlations. The feature importance remains consistently high, around 3.5, when R^2 is between 0 and 0.97, but abruptly falls to -14 when R^2 is above 0.97. The model also suggests that a DDoS attack is less likely if the coefficient of determination exceeds 0.97. This observation aligns with basic cybersecurity principles. During DoS, DDoS flooding, the correlation between incoming and outgoing packet sizes may decline as the network becomes inundated with unwanted packets[10].

The second plot on the right in Figure 1 displays the feature importance for Recon and Spoofing attacks. Neglecting the outliers, the peak feature importance of 1.1 for Recon appears when R^2 lies between 0.9 and 0.97. Recon attacks typically involve gathering information about a target system, often as a precursor to a more direct assault. An adept attacker might endeavor to imitate the standard packet size distribution of the network to evade detection[9]. This scenario explains why the feature importance escalates as the coefficient of determination approaches 1. In the case of Spoofing, the feature importance score ranges between -4 and -2 when the R^2 value is between 0.9 and 0.97. The correlation between incoming and outgoing packet sizes is not apparent in such attacks. The machine learning model's generated negative feature importance aligns with this observation. In general, R^2 serves as a strong indicator for the machine learning model to distinguish DDoS, DoS, and Mirai attacks from Recon, Spoofing, and benign traffic.

4.3 TCP Flags (ACK flag and SYN flag)

Transmission Control Protocol (TCP) flags, despite their eight different types, are not all reliable indicators of attacks. Upon analyzing their influence on our model's accuracy, we discovered that the acknowledgment (ACK) and Synchronisation (SYN) flags played a more significant role. The SYN flag, typically used to establish connections, can also be exploited in certain forms of cyberattacks. The third plot on the right in Figure 1 presents the feature importance of the SYN flag for each attack. As a categorical variable, the SYN flags are either 0 for unset (on the left of the plot) or 1 for set (on the right). For the Mirai attack, a set SYN flag signifies a strong negative feature importance, approximately -28. The SYN flag's relevance to Mirai hinges on the types of DDoS attacks the Mirai botnet implements. Our dataset solely contains three forms of Mirai attacks - GRE IP Flood, GRE Ethernet Flood, and UDP Plain attacks. These attacks operate on the Generic Routing Encapsulation (GRE) and User Datagram Protocol (UDP), which belong to a different layer than TCP and do not utilize the SYN flag. Additionally, we observe that Benign and Spoofing also show negative feature importance, with scores of -4.5 and -2.7, respectively. Similar to Mirai, neither Address Resolution Protocol (ARP) nor Domain Name Service (DNS) Spoofing involves the SYN flag. In terms of benign traffic, a high count of SYN packets might suggest malicious activity. The negative feature importance for Benign is likely due to the interaction the model identified between the SYN and ACK flags.

The fourth plot on the right in Figure 1 reveals the feature importance of the total ACK count for each class. Unlike the former feature, the ACK count measures the total number of ACK flags in a flow and is thus a continuous variable. The ACK flag is predominantly used in benign traffic to signal the acknowledgment of packet receipt. Both the fourth and fifth graphs demonstrate a positive correlation between feature importance and benign traffic. During a DDoS attack, the attacker inundates the target's network with excessive internet traffic. Yet, DDoS attacks frequently involve a large quantity of spoofed SYN packets as opposed to ACK packets. A negative feature importance for a DDoS attack is expected for high values of ACK count. As mentioned earlier, DoS and Mirai attacks bear similarities to DDoS attacks; thus, these types of attacks exhibit similar patterns of feature importance[2]. Regarding Recon attacks, ACK flags can be employed in various network scans, such as TCP ACK scanning. The plot also reveals a positive correlation between the ACK flag feature importance and Recon attacks. Lastly, given that our dataset does not include instances of ACK Spoofing, we anticipate a negative feature importance for Spoofing.

In combination with R^2, the SYN and ACK flags allow the machine learning model to classify attacks more effectively. Based on previous features, the model can segregate classes into two groups: DDoS, DoS, Mirai, and Recon versus Spoofing and Benign. A set SYN flag eradicates the possibility of Mirai in the first group and reduces the likelihood of Spoofing and Benign in the second group. The ACK count assists in identifying DDoS and Recon attacks while diminishing the chance of Spoofing.

4.4 Discussion

While the EBM yields an impressive accuracy rate of 99% and an F1 score above 90%, our research still has several limitations and offers potential areas for improvement. The dataset encompasses 105 IoT devices, which, although considerable, might only be representative of specific types or categories of IoT devices encountered in real-world scenarios. Furthermore, it includes only specific instances of DoS, DDoS, Spoofing, Recon, and Mirai attacks. The omission of other types of attacks from the dataset, such as Web-based and BruteForce, could constrain the scope of cybersecurity threats that our research is capable of addressing.

Additionally, the original dataset does not proportionally represent the individual methods employed in each attack, which could affect the feature importance's reliability. For instance, it houses 7 million instances of Internet Control Message Protocol (ICMP) Flood for DDoS attacks, contrasted with a scant 29 thousand instances of Hypertext Transfer Protocol (HTTP) Flood. This unequal representation may skew the EBM's feature importance, as it tends to lean towards the majority group. We have ascertained that the specific characteristics of the dataset heavily influence the predictions made by the EBM. For example, when examining the Mirai attacks, one feature, the minimum packet length, has its feature importance peak at 501 to 591 bytes (the fourth plot on the left in Figure 1). This peak is primarily because our dataset only includes three types of Mirai attacks—GRE IP Flood, Greeth Flood, and UDP Plain—which all display packet sizes within the 554 to 592 bytes range[11]. Other variants of Mirai attacks, like the SYN Mirai attack, which has a packet size of merely 74 bytes[11], are absent from our dataset. As a result, the IoT analysis we suggested, based on feature importance, is limited to the attack types included in our dataset. Encountering new attack types may significantly undermine the machine learning model's accuracy rate. Future research should aim to obtain a sufficiently comprehensive dataset, featuring the most prevalent internet attack forms while maintaining a balanced representation for each class.

It's also important to note that a machine learning-based Intrusion Detection System (IDS) for IoT should be deployed alongside other security measures, such as encryption, authentication, and access control mechanisms, to provide a comprehensive security solution.

5 Conclusions and Future Scope

The utility of machine learning-based intrusion detection systems, while promising, requires careful implementation and evaluation. In our study, we trained an EBM on an extensive IoT dataset to identify potential attacks. Through an analysis of feature importance, we were able to interpret the decision-making process of the machine learning model. Our validation process generally agreed that the predictions made by the machine learning model align with fundamental cybersecurity principles. However, there were instances where the model's predictions were challenging to explain or did not conform with these principles. Furthermore, we identified limitations regarding the model's predictions. Our analysis revealed that the machine learning model's predictions were strongly tied to the specific characteristics of the training IoT dataset, thereby raising concerns about the model's reliability when applied to real-world attack detection.

As we move forward in developing machine learning-based intrusion detection systems, it is vital to ensure the training dataset meets three critical criteria. Firstly, the network topology for IoT devices should be comprehensive, mimicking real IoT operations closely. Secondly, the dataset should encompass a wide variety of attack types, utilizing all prevalent tools and frameworks employed to execute these attacks. Lastly, the dataset should be balanced not only across different attack types but also across the various instances of methods used to execute such attacks. Despite satisfying these criteria, it is important to understand the limitations of these models in real-world applications. The system should ideally be applied only to a similar set of IoT devices as those in the training set, and it should be recognized that attacks not included in the training set may not be detected. By addressing these factors, we believe the utility and reliability of machine learning-based intrusion detection systems for IoT can be significantly improved.

References

[1] Adel Abusitta et al. “Deep learning-enabled anomaly detection for IoT systems”. In: *Internet of Things* 21 (2023), p. 100656.

[2] Manos Antonakakis et al. “Understanding the mirai botnet”. In: *26th USENIX security symposium (USENIX Security 17)*. 2017, p. 1095.

[3] Mohamed Amine Ferrag et al. “Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning”. In: *IEEE Access* 10 (2022), p. 40303.

[4] Hanan Hindy et al. “Machine learning based iot intrusion detection system”. In: *arXiv preprint arXiv:2006.15340* (2020), pp. 73–84.

[5] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017).

[6] Achmad Akbar Megantara and Tohari Ahmad. “Feature importance ranking for increasing performance of intrusion detection system”. In: *2020 3rd International Conference on Computer and Informatics Engineering (IC2IE)*. IEEE. 2020, pp. 37–42.

[7] Euclides Carlos Pinto Neto et al. “CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment”. In: (2023).

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

[9] Sirikarn Pukkawanna, Youki Kadobayashi, and Suguru Yamaguchi. “Network-based mimicry anomaly detection using divergence measures”. In: *2015 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE. 2015, p. 1.

[10] Zhongmin Wang and Xinsheng Wang. “DDoS attack detection algorithm based on the correlation of IP address analysis”. In: *2011 International Conference on Electrical and Control Engineering*. IEEE. 2011, p. 2951.

[11] Ron Winward. *Iot attack handbook*. 2018. URL: `https://falksangdata.no/wp-content/uploads/2021/04/Mirai-Handbook.pdf`.

[12] M Zolanvari et al. “WUSTL-IIOT-2O2l Dataset for IIoT Cybersecurity Research”. In: *Washington University in St. Louis, USA* (2021).

Inherent Flaws of Modbus: A Case Study on the Absence of Authentication and Authorization in a Garage Door System's Programmable Logic Controller*

Tristen Greene and Thomas R. Walsh
Computer Science & Electrical Engineering
Eastern Washington University
Spokane, WA 99202
[tgreene9, twalsh]@ewu.edu

Abstract

In the era of the Industrial Internet of Things (IIoT), Programmable Logic Controllers (PLCs) are pivotal in diverse industrial processes, including assembly lines and power grids. Despite their significance, the security of these devices is increasingly under examination due to vulnerabilities to cyberattacks. This paper explores a specific vulnerability within PLCs, focusing on a case study examining a PLC network's exploitation in a garage door controller system. By illustrating a cyberattack and emphasizing the absence of authentication and authorization in the widely-used Modbus protocol for PLC systems, the research team underscores a pronounced weakness in PLCs. Using NMAP, an open-source network scanning tool, and the PyModbus Python library for read and write operations on Modbus memory, this study emphasizes the urgent need for enhanced security protocols in PLC systems and aims to spur further academic research in this field.

*Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

In the digital era, Programmable Logic Controllers (PLCs) are indispensable in various industrial applications, controlling diverse functions from assembly line operations to intricate manufacturing processes. Their critical roles in vital infrastructures, such as energy distribution and water treatment plants, emphasize their importance. This significance, however, also makes them prime targets for cyber adversaries.

Created in the 1960s, PLCs were developed at a time when cyber threats were not a primary concern. Though strides have been made to address vulnerabilities, inherent weaknesses still persist, like CVE-2022-38773[3], enabling malicious actors to bypass protective boot features, allowing unauthorized modifications to operating code and data.

Alarmingly, many systems lack robust protective measures. This gap often arises from the challenges faced by engineers when incorporating security solutions into legacy systems, leaving many PLCs vulnerable to breaches.

This paper delves into an inherent flaw in the Modbus protocol, particularly its absence of authentication and authorization. Using a garage door controller system as a case study, the research team showcases how PLC networks can easily be exploited due to this fundamental weakness. By highlighting this vulnerability, the study seeks to foster further research and innovation in PLC security.

2 Background

2.1 Programmable Logic Controllers and the Modbus Protocol

Programmable Logic Controllers (PLCs) are essential for industrial control systems, engineered to engage sensor data or inputs, process them, and generate designated outputs based on predefined logic. The Modbus protocol, a universal language that almost all industrial devices can utilize for integration, is a critical aspect of PLC communication.

Initially developed in the late 1970s for PLC communication, Modbus employs a straightforward master/slave communication methodology. Its variant, Modbus TCP, was developed in the late 1990s and has achieved industry-standard status in numerous sectors. Modbus TCP/IP embeds the Modbus protocol within a TCP/IP network, facilitating Modbus communication over modern Ethernet networks. This particular variant has gained significant popularity, primarily due to the broad adoption of Ethernet networks in industrial automation.

At the core of Modbus TCP/IP are several fundamental elements that facilitate its operation. The protocol encapsulates Modbus messages within

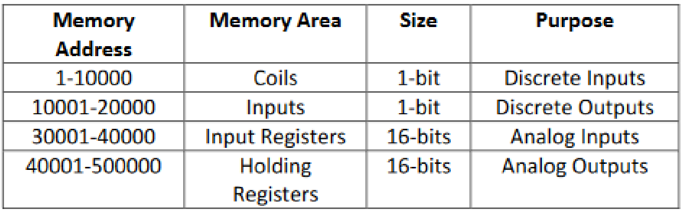
TCP/IP networks, a significant shift from the original serial (RS-232 or RS-485) networks. This encapsulation allows for communication over longer distances and among a larger number of devices. Adhering to its roots, Modbus TCP/IP employs a client/server architecture in which a single device, acting as the client, initiates communication, while other devices, acting as servers, respond.

Communication in Modbus TCP/IP typically occurs over port 502, providing a standardized avenue for data transmission. While efficient and robust, the protocol's inherent limitations, such as lack of built-in security features and unencrypted data transmission, can pose risks for specific applications.

2.2 The Modbus Memory Map

PLCs and other industrial automation devices typically have on-board memory crucial for programming. Inputs are often buffered, or stored, in memory before logic is applied, enhancing the adaptability and maintainability of the PLC's program.

Understanding a typical PLC's memory map, especially when using the Modbus protocol, is essential. The memory map is neatly segmented into distinct areas, each serving a specific function. The four primary sections, namely Coils (1-10000), Discrete Inputs (10001-20000), Input Registers (30001-40000), and Holding Registers (40001-50000), hold unique functionalities. Coils manage binary outputs that can be set or reset by the device. Discrete Inputs are read-only binary inputs. Input Registers, typically housing analog inputs, are used for reading data from devices. Lastly, Holding Registers are utilized for both reading and writing data, including output control and configuration changes.



Memory Address	Memory Area	Size	Purpose
1-10000	Coils	1-bit	Discrete Inputs
10001-20000	Inputs	1-bit	Discrete Outputs
30001-40000	Input Registers	16-bits	Analog Inputs
40001-500000	Holding Registers	16-bits	Analog Outputs

Figure 1: The Modbus Protocol Memory Map

While this structure fosters an efficient programming process, it also presents a potential attack surface for cyber threats, which can exploit these stored variables within the memory. For example, buffer overflows might allow an attacker to modify register values or cause the device to overflow its buffer, potentially rendering it unresponsive[2]. Securing these memory spaces against various forms of exploitation is a vital aspect of industrial cybersecurity.

2.3 Ladder Logic

Ladder Logic, a graphical programming language for PLCs, evolved from relay circuit diagrams. A common paradigm is the "Seal-In" circuit, which maintains a device's state after the initial input is disengaged, useful for devices like garage door controllers, as depicted in Figure 2. This function is often employed in PLC programming due to its ability to control devices that need to maintain their operating state without continuous input, such as motors, lights, or, in our case of interest, a garage door.

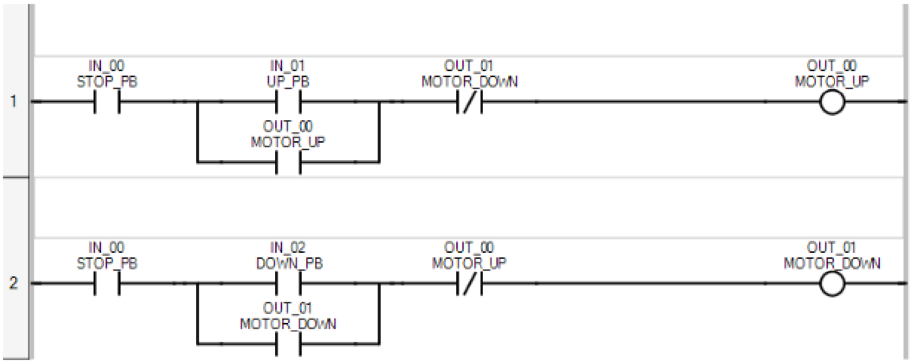


Figure 2: Example of Simple Seal-in Circuit in Garage Door Controller

For instance, when a garage door opener is triggered, a seal-in circuit allows the motor to continue operating until the door is fully open, even after the user releases the button. Similarly, the door will stay open until another input tells the PLC to close it. The seal-in circuit thus offers a way to control devices in an efficient and user-friendly manner.

However, the very feature that makes seal-in circuits advantageous also presents a potential risk. If a threat actor gains access to the PLC controlling the garage door, they could potentially manipulate the seal-in variable that controls the motor. By setting this variable to 'true', the attacker could force the door to open, regardless of the intended state. This kind of intrusion underscores the need for robust PLC security.

2.4 History of Programmable Logic Controllers Exploitation

The Stuxnet worm is a well-documented example of how threat actors can exploit Programmable Logic Controllers (PLCs) to cause significant disruption. Stuxnet was a sophisticated piece of malware targeting Siemens Step7 software, commonly used for programming PLCs. It was designed to modify the code of

PLCs controlling centrifuges that were used for uranium enrichment, causing them to spin out of control while simultaneously returning normal readings to the monitoring systems. Stuxnet was launched in 2009 and exposed in 2010, and it was the first known case of a cyber weapon designed to cause real-world physical damage[4]. As such, it was a wake-up call to the cybersecurity community regarding the potential in the field of PLC exploitation.

3 The Activity

3.1 The Scenario

Focusing on exploiting Programmable Logic Controllers (PLCs), essential components in modern industrial applications, our methodology began within the PLC network context. In real-world scenarios, the initial step typically involves accessing the network. Once access has been gained, further attacks can be performed on critical system components like PLCs.

3.2 Identifying Programmable Logic Controllers on the Network

Upon unauthorized network access, a threat actor aims to map interconnected devices and identify potential industrial system controllers, particularly PLCs. The Modbus protocol is commonly used, and tools like NMAP are invaluable for this purpose. With Python-NMAP providing programming-oriented capabilities, it was chosen for its efficiency and compatibility with the task[1].

3.3 Scanning for Modbus Devices

To identify devices with an open port 502, a significant port for Modbus communication, we used the python-nmap package. The following pseudocode details the logic:

```
Data: Network range
Result: List of devices with open port 502
Initialize PortScanner as nm;
Scan network for port 502 using nm;
Initialize empty list clientsWithPort502Open;
for each host in scanned hosts do
    if port 502 is open on host then
        Add host to clientsWithPort502Open;
return clientsWithPort502Open;
```

Algorithm 1: Pseudocode for Identifying devices with open port 502

The research team utilized this pseudocode to detect devices with an open port 502. The network range encapsulates the IP address and subnet mask of the compromised machine. This information is gleaned using the netifaces module, a Python library tailored for extracting network interface details. The function's output is a list of devices with the Modbus port open.

3.4 Profiling Detected Devices

The subsequent objective is to determine the nature of each device at these addresses. Major PLC manufacturers such as Siemens and Allen-Bradley employ dedicated function codes for retrieving device information. By utilizing these function codes, the research team was able to discern the type of device associated with each address in the list, thereby gaining a more comprehensive understanding of the network's structure. This profiling process involves careful analysis of the device responses, cross-referencing them with known manufacturer specifications, and categorizing them based on their functionalities and roles within the network.

3.5 Reading of the Modbus Memory Map

Once the PLCs on the network have been identified, the next phase involves reading the Modbus memory map of these devices. This process requires specific function codes to communicate with the PLCs via the Modbus protocol. The research team developed a framework designed to read the Modbus memory map, utilizing specific function codes compatible with different memory sections, such as coils, discrete inputs, holding registers, and input registers. The reading operation involves the transmission of specific requests to the server, allowing for the delineation of the memory structure. This process not only establishes the foundation for subsequent analysis and emulation but also enables the extraction of vital information about the PLC's configurations and functionalities.

3.6 PyModbus and Integration of Real-world Apparatus

The research team leveraged PyModbus, a Python library that offers Modbus functionality, to create the reading process. This library facilitated communication with PLC simulators and real-world PLCs such as a garage door controller, generously supplied by a fellow student for their project. By using PyModbus in conjunction with simulators, the team was able to generate and validate the mapping of the Modbus memory map. This approach ensured the precision of the data, with simulated values used to authenticate the mapping's accuracy. The actual values within the PLC's Modbus memory map managing the garage door controller matched those used in the corresponding ladder logic program. The logic for the reading operation can be summarized with the following pseudocode:

```
Data: Client, Optional Addresses
Result: Memory Map with Read Values
Initialize sections with configurations for coils, discrete inputs, holding
  registers, and input registers;
Initialize empty dictionary memory_map;
for each (section, config) in sections do
    Initialize start_address from config;
    Initialize num_elements from config;
    Initialize read_func from config;
    Initialize empty dictionary values;
    Create address_range from start_address to start_address +
      num_elements;
    if addresses is provided then
        Filter address_range to only include provided addresses;
    for each address in address_range do
        response = call read_func with address;
        if response is not an error then
            if section is coils or discrete_inputs then
                Add response bit to values with key as address;
            else
                Add response register to values with key as address;
        Continue to next address;
    if values is not empty then
        Add values to memory_map with key as section;
```

Algorithm 2: Reading Modbus Memory Map

3.7 Insights Derived from the Memory Map Analysis

A detailed analysis of the memory map could reveal the configurational characteristics of the PLCs. Armed with knowledge in PLC programming, a threat actor could deduce the specific functionalities and processes governed by the PLC within an industrial control system network. Alarmingly, the same absence of authorization that permitted the research team to send read commands to the PLC can now be exploited to write to the Modbus memory. Specifically, the coils and holding registers, allow for write operations, enabling the control of discrete outputs and analog outputs.

3.8 Writing to the Modbus Memory Map

After successfully reading the Modbus memory map, the research team began the last phase, which involves writing to the PLCs using the Modbus protocol. Specific functions were crafted to handle the writing to different memory sections, such as coils and holding registers. This process includes selecting the appropriate write function based on the targeted section, followed by transmitting the desired value to the corresponding address on the PLC. The writing operation allows for controlled modification and interaction with the PLC's memory structure. It also provides a mechanism to emulate and understand potential manipulations that might be executed by malicious entities. The ability to write to the Modbus memory map, coupled with memory map updating, strengthens the analysis by extending the scope of understanding and engagement with the PLC's behavior, configurations, and potential vulnerabilities. In the context of our research, we specifically focused on writing to the occupied coils in the Modbus memory map of a garage door controller.

The logic for the writing operation can be summarized with the following pseudocode:

```
Data: Client, Section Name, Address, Value
Result: Success or Failure of the Write Operation
sections = [("Coil", write_coil_function), ("Holding Register",
 write_register_function)];
for each (valid_section_name, write_func) in sections do
    if valid_section_name equals section_name then
        response = call write_func with Address and Value;
        if response is an instance of ModbusException then
            Log "Failed to write to section_name at address";
            return False;
        else
            Update the memory map with section_name and address;
            return True;
    Log "Failed to write to section_name at address due to exception";
    return False;
Log: "Invalid section name";
return False;
```

Algorithm 3: Writing to Modbus Memory Map

3.9 Successful Modification

By writing accurately to the 'motor up' variable in the Modbus memory map, the research team altered the state of the PLC, triggering the 'motor up' operation in the garage door controller. This activity resulted in the successful opening of the garage door, demonstrating the potential impact of such intrusions. The demonstration underscores the far-reaching consequences of these vulnerabilities in real-world scenarios.

Writing to the Modbus memory map allows an attacker to manipulate the behavior of the PLC, potentially leading to unauthorized or unintended actions. The ability to control the state of the PLC's outputs through the Modbus protocol poses significant risks in industrial systems. Depending on the application, unauthorized access to and manipulation of the Modbus memory map could result in physical damage, safety hazards, or operational disruptions.

The successful alteration of the garage door controller's behavior highlights the importance of robust security measures for industrial control systems, including PLCs. It stresses the need for adequate access controls, authentication mechanisms, and intrusion detection systems to prevent unauthorized write operations on critical Modbus memory addresses.

4 Follow-up Work

In the light of the findings from this study, there are various potential avenues for future research. They include:

4.1 Exploration of Segmented Network Infrastructures

This research was conducted within a "flat" network infrastructure, where Information Technology (IT) and Operational Technology (OT) networks were not segregated. Real-world industrial networks often employ a Demilitarized Zone (DMZ) to separate IT and OT networks, adding an extra layer of security. Exploring this structure is crucial to understanding security in complex industrial environments. Future research could explore hacking techniques that could potentially breach this security measure, thus expanding the understanding of potential vulnerabilities in these environments.

4.2 Mitigation Strategies

As vulnerabilities and potential threats are identified, an important future work area involves the development of robust mitigation strategies. This can include techniques such as advanced firewall configurations, intrusion detection systems, and regular security audits. Research in this area will play a crucial role in maintaining the security of industrial systems in the face of evolving cyber threats.

5 Concluding Remarks

The rapid progression of cyber threats accentuates the necessity for fortified and secure industrial control systems. This research underscores that Programmable Logic Controllers (PLCs), despite their ubiquity in industrial settings, remain susceptible to cyber intrusions. By spotlighting the lack of authentication and authorization in Modbus protocol, this study offers a deeper understanding of the challenges in industrial cybersecurity.

The findings from this investigation are pivotal for grasping the existing gaps in Authentication and Authorization mechanisms within many systems. They also serve as a foundation for the conceptualization and creation of more fortified systems in the future. We anticipate that this exploration will catalyze further academic and practical endeavors in the realm of industrial control system security. As our dependence on these systems intensifies across various sectors, the imperative to shield them from potential cyber threats becomes even more critical.

6 Acknowledgements

We would like to thank Dr. Stuart Steiner for providing all the PLC equipment used in this project through grants he received.

References

[1] Gurline Kaur and Navjot Kaur. “Penetration testing–reconnaissance with NMAP tool”. In: *International Journal of Advanced Research in Computer Science* 8.3 (2017), pp. 844–846.

[2] Parul Sindhwad and Faruk Kazi. “Exploiting Control Device Vulnerabilities: Attacking Cyber-Physical Water System”. In: *2022 32nd Conference of Open Innovations Association (FRUCT)*. IEEE. 2022, pp. 270–279.

[3] Yuanzhe Wu, Grant Skipper, and Ang Cui. “Uprooting Trust: Learnings from an Unpatchable Hardware Root-of-Trust Vulnerability in Siemens S7-1500 PLCs”. In: *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE Computer Society. 2023, pp. 179–190.

[4] Kim Zetter. *Countdown to Zero Day: Stuxnet and the launch of the world’s first Digital Weapon*. Crown Publishers, 2016.

Pilot Project to Study the Effectiveness of Job Interview Preparation Workshops*

Ayat Hatem
Computer Science Department
California State University Stanislaus
Turlock, CA 95382
ahatem@csustan.edu

Abstract

The demand for diversity in the tech industry has been increasing. However, students from underrepresented minorities and first generation college students are usually under-prepared and do not have the essential resources, including time and information, needed to excel in technical interviews. In this study, we propose a four-week job interview preparation workshop to equip students with the resources needed to succeed in job interviews. The workshop briefly covers the different aspects of a job interview, such as how to tackle different questions, effectively communicate your solution, and write your code on a whiteboard. The study's results demonstrate that students responded positively to the workshop, reporting reduced anxiety and enhanced preparedness for technical interviews. We describe in details the format of the workshop and discuss how it can be further modified to accommodate more students.

1 Introduction

In order to land in a technical job, most companies ask for an intrusive technical interview [5]. Applicants are asked different types of questions that they need to correctly answer, including real time coding and fundamental knowledge in

*Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

computing questions. They are expected to provide an answer for the given questions while speaking through their thought process, which proves to be mentally exhausting [1].

Students are expected to prepare months in advance. However, the cultural experiences of underrepresented minorities impact their availability to prepare for the technical interviews in comparison to white students [10]. Taking care of the family, having health issues, and working are among the factors affecting the students' availability. There are many promising programs and books available to the students [2, 12]. However, students do not usually know about those programs and they do not understand their effectiveness, especially that they require a long commitment with family and other work requirements [7].

Students that landed a job, after going through many job interviews, suggested that campuses need to provide more support to students to make them more prepared to job interviews [11]. The support can be in the form of providing mock interviews, better advertisements of any opportunities offered through the different clubs, helping in developing their communication skills, and even creating core or elective courses to help students prepare for the hiring process. However, many minorities serving institutions lack resources, hence, limiting what faculty can do to support their students [13].

To tackle the above issues, in this work, we are proposing a four weeks coding interview preparedness workshop sessions to introduce students to the different aspects of coding interviews and how to prepare for them. The goal of the workshop is to increase the familiarity of the students with the interview process and encourage them to prepare early for the job interviews. It will not require a lot of their time and can be offered virtual or in-person. The students included in this study are all from California State University Stanislaus, which is a Hispanic Serving Institution (HSI). However, all students in minority serving institutions face the same challenges and the idea of the workshop can be used by different institutions to support their students.

2 Related Work

Kapoor et al introduced the idea of adding interview preparation activities to data structures and algorithm class [8, 9]. They added internships discussions, mock interview exercises, and short graded programming assignments. The study included 363 students with 257 as the study group and 106 as the control group. Students reported that the activities increased their awareness and preparedness for interviews. Even though the study is important, it included a large group of students with no direct faculty-students interaction. That lacks more individualized feedback and more of general feedback given to all students. Additionally, even though the idea is effective, it takes from the time assigned to learn the main materials in the data structures and algorithms course, thus requiring either attending extra sessions or taking time assigned to study the course materials itself.

Dillon et al studied the effectiveness of exposing early CS majors to coding interview practices using white boarding techniques [4]. Unlike Kapoor et al, The goal of this study was understanding the effect of the exposure rather than preparing students for job interviews. It showed that students exhibited anxiety when exposed to those practices.

Griffin et al focused on redesigning and co-developing some courses by industry professional and CS Professors [6]. They co-taught the courses for an academic semester. Additionally, the industry partners provided volunteers to carry mock interviews, assess as mentors, and serve as guest speakers and panelists. The study showed students gained a lot of experiences and were more prepared for technical interviews and their cs career. Even though some aspects of the study can be applied, it is not feasible for some campuses to redesign courses or offer new ones with the existing limited resources [13] .

3 Workshop Format

Students need to tackle four different aspects to be prepared for job interviews: communication skills and how to effectively convey solutions, decent knowledge of data structures and algorithms, how to solve coding problem in a white board, and different problem solving techniques [12]. We designed the workshop to briefly address the mentioned aspects and direct the students to further study by themselves to become better prepared.

The one-hour-long workshop sessions are designed for CS major students who have completed data structure and algorithm courses or are currently enrolled in one. Students from introduction to programming courses are welcome to attend the workshop as listeners, rather than active participants. The primary emphasis of the workshop is to accommodate a small cohort of 2-4 students per group, collaborating with a faculty member, hence allowing for a more personalized feedback and addressing individual student concerns.

The four weeks start with a discussion about the interview process in the first week and end with a mock interview in the last week. The second and third weeks focus on honing problem solving techniques. Meanwhile, students engage in hands-on practice of various interview skills, including communication and whiteboard utilization, through solving problems from "Cracking the Coding Interview" book by McDowell [12].

At the conclusion of each session, every student receives a comprehensive and personalized feedback highlighting areas for improvement. Moreover, we actively encourage students to provide constructive feedback themselves. This two-way feedback process not only facilitates individual growth but also fosters a collaborative and supportive learning environment.

For the mock interview, students provide their own question, with each student alternating in role between acting as an interviewer, applying effective interview skills, or as interviewee. After finishing the workshop, students receive a copy of the book to further enhance their preparation for the interview process.

4 Students Experiences

The workshop was provided for four different cohorts spanning three different semesters. Two of the cohorts were in-person while the other two were online through zoom and using google docs. Each cohort consisted of either two or three students, with a total of 10 students attending the workshop so far. Students were asked to fill out a survey regarding their experience with the workshop. The results here are based on the survey answers filled out by six students.

The survey focused on three main aspects: previous coding interview experience, job interview anxiety before and after the workshop, and knowledge gained after attending the workshop. Out of the six students, two indicated that they had virtual job interviews before attending the workshop. However, none of the students experienced any in-person interviews (Figure 1).

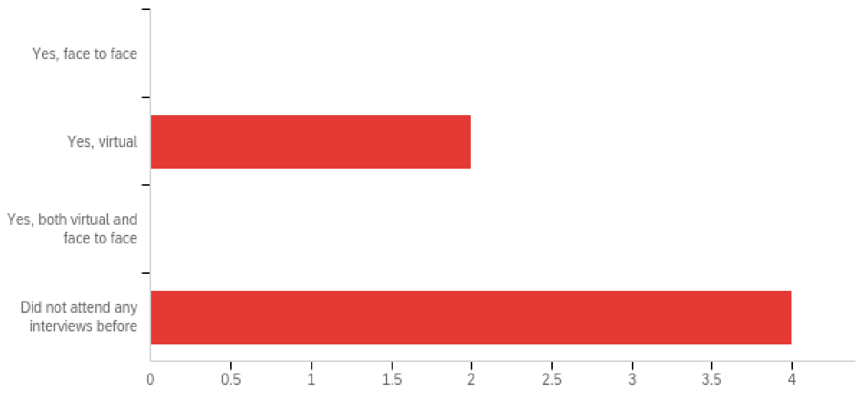


Figure 1: Job interview experience before attending the workshop.

All students indicated that they had a high anxiety level regarding job interviews. However, their anxiety level improved after attending the workshop, including the two students who had virtual interviews before (Figure 2). As

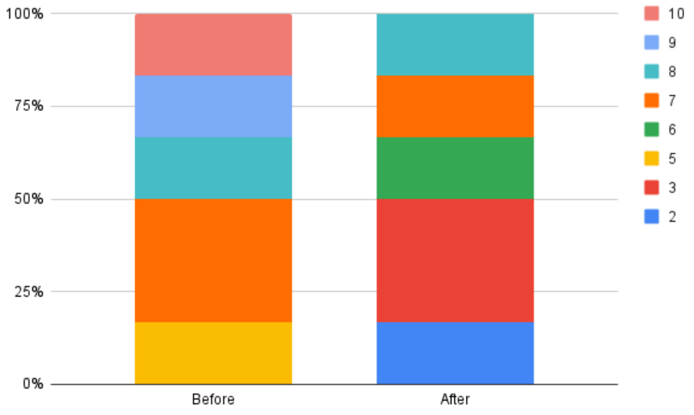


Figure 2: Anxiety level before and after attending the workshop.

Dillon et al discussed, students exhibit a high anxiety level regarding job interviews [4]. Our results indicate that even a four week workshop could improve their anxiety level.

Students were also satisfied with the setup of the workshop and the materials they learned (Figure 3). Additionally, in a 10-point Likert scale, they would recommend other students to attend the workshop (a mean of 9 and std deviation of 1.15) .

5 Conclusion & Future Work

A lot of minority serving institutions suffer from lack of resources [8]. Adding a complete course or interview preparation materials to existing courses could constraint both faculty and students alike due to the limitation of both resources and time. To find a middle ground, in this work, we proposed a four week workshop to introduce students to technical interviews and equip them with resources needed to make them better prepared. Even though students will still need to prepare more for the job interviews after finishing the workshop, we wanted to understand if the workshop at least help students to find the start point and move forward on their own to prepare for job interviews. Students had a positive attitude regarding the workshop and would recommend it to other students.

However, One of the drawbacks of the current workshop format is that it does not allow working with a large group of students, hence, not every student will gain the same knowledge. Additionally, some students refrained

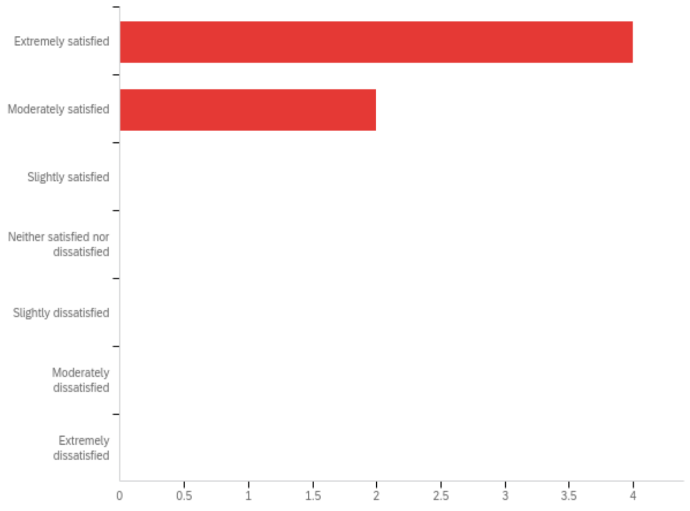


Figure 3: Students' satisfaction level with the workshop and the materials covered.

from signing up as there was no credit points associated with the workshop. One way to tackle this issue would be to better advertise for the workshop or add guest speakers that went through technical interviews before. Additionally, we can engage local alumni as interview coaches [14, 3].

Some students recommended as well adding more sessions to practice more problems and mock interviews and cover other types of problem solving techniques. Hence, as a future work, we could study if it is feasible to offer a one credit semester long course where students meet for one hour weekly. There are advantages of adding a one credit course over adding more materials to existing courses, including not taking time from the required core courses and students receiving credits towards their graduation. On the other hand, this will require allocating more resources, such as faculty and students assistants. Other future work will be to follow up with students who attended the workshop and see if they applied for job interviews and if they believe the workshop helped them to become better prepared.

References

[1] Mahnaz Behroozi et al. "Dazed: measuring the cognitive load of solving technical interview problems at the whiteboard". In: *Proceedings of the 40th Interna-*

tional Conference on Software Engineering: New Ideas and Emerging Results. 2018, pp. 93–96.

[2] *Computing talent initiative.* `https://computingtalentinitiative.org/accelerate/`.

[3] 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. Minneapolis, MN, USA: Association for Computing Machinery, 2019, pp. 29–35. ISBN: 9781450358903. DOI: `10.1145/3287324.3287508`. URL: `https://doi.org/10.1145/3287324.3287508`.

[4] Edward Dillon et al. "Exposing Early CS Majors to Coding Interview Practices: An HBCU Case Study". In: *2021 Conference on Research in Equitable and Sustained Participation in Engineering, Computing, and Technology (RESPECT).* IEEE. 2021, pp. 1–4.

[5] William Gant and William Gant. "Why software development interviews are hard". In: *Surviving the Whiteboard Interview: A Developer's Guide to Using Soft Skills to Get Hired* (2019), pp. 1–6.

[6] Jean Griffin et al. "Innovative Courses that Broaden Awareness of CS Careers and Prepare Students for Technical Interviews". In: *Journal of Computing Sciences in Colleges* 38.5 (2022), pp. 54–64.

[7] Amanpreet Kapoor and Christina Gardner-McCune. "Barriers to securing industry internships in computing". In: *Proceedings of the Twenty-Second Australasian Computing Education Conference.* 2020, pp. 142–151.

[8] Amanpreet Kapoor and Christina Gardner-McCune. "Introducing a Technical Interview Preparation Activity in a Data Structures and Algorithms Course". In: *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2.* 2021, pp. 633–634.

[9] Amanpreet Kapoor, Sajani Panchal, and Christina Gardner-McCune. "Implementation and Evaluation of Technical Interview Preparation Activities in a Data Structures and Algorithms Course". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 2023, pp. 882–888.

[10] Stephanie J Lunn. "Uneven playing field: Examining preparation for technical interviews in computing and the role of cultural experiences". In: *2021 ASEE Virtual Annual Conference Content Access.* 2021.

[11] Stephanie Jill Lunn, Ellen Zerbe, and Monique S Ross. "Need for Change: How Interview Preparation and the Hiring Process in Computing Can Be Made More Equitable". In: *2022 CoNECD (Collaborative Network for Engineering & Computing Diversity).* 2022.

[12] Gayle Laakmann McDowell. *Cracking the coding interview: 189 programming questions and solutions.* CareerCup, LLC, 2015.

[13] "Special Challenges for Minority-Serving Institutions". In: *Retention in Computer Science Undergraduate Programs in the U.S.: Data Challenges and Promising Interventions.* New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450388320.

[14] Tammy VanDeGrift. "Alumni as Teachers and Mentors for CS 1 Students: Solving the Staffing Shortage and Students' Reflections about Career and College Advice". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 1124–1130. ISBN: 9781450394314. DOI: `10.1145/3545945.3569721`. URL: `https://doi.org/10.1145/3545945.3569721`.

Exploring Performance Gaps by Gender, Ethnicity and Teaching Modality in CS3[*]

Elena Harris
Computer Science Department
California State University, Chico
Chico, CA 95929
eyharris@csuchico.edu

Abstract

Programming skills are an essential component of computer science education. CS3, a third-level programming course, is especially challenging for students, with a "D, F, withdraw" (DFW) rate of 20% to 28% for students in majors for which this course is required. Here, performance gaps in the three assessment areas – a Programming Exam, Written Test and Final Exam – were analyzed for three types of student groups: by gender, by ethnicity and by teaching modality (in-person vs online). The performance data from eight semesters was used for this analysis. Students taking CS3 in person statistically outperformed students taking CS3 online in all performance categories. In a Programming Exam, there is a statistically significant performance gap between top performers (all problems solved correctly on all Programming Exams) by gender and by ethnicity. There is a statistically significant difference in performance on a Written Test between in-person students who took a Tutor-Retake Written Test with up to three attempts and in-person students who were allowed only a single attempt per test.

1 Introduction

The effective teaching of computer science programming courses requires continuous refinement of teaching and assessment strategies to boost student mo-

[*]Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

tivation and improve performance [5]. Moreover, there is an ongoing effort to balance gender and racial representations in Computer Science (CS) courses [3]. Improvements of assessment strategies can foster inclusiveness, equity, and diversity among students. This records-based project is a retrospective study that investigates whether two assessment tools, a Programming Exam and a Written Test, are associated with the performance gap between well-represented and under-represented computer science students in a CS3 course, the third level computer science programming course. This is a preliminary study that can shed light on the need for new strategies in designing Written and Programming Exams for a CS3 course.

This study aims to answer the following research questions:

> **Research Question 1**: Is there a statistically significant difference between well-represented and under-represented ethnicity groups in performance on the Programming Exams/Written Exam/Final Exam/Overall Grade.
>
> **Research Question 2**: Is there a statistically significant difference between well-represented and under-represented gender groups in performance on the Programming Exams/Written Exam/Final Exam/Overall Grade.
>
> **Research Question 3**: Is there a statistically significant difference between Online- and In-Person-taught groups in performance on the Programming Exams/Written Exam/Final Exam/Overall Grade.

The objectives and contributions of this study involve evaluating the effectiveness of a Programming Exam and a Written Test; identifying possible performance gaps between student groups by gender, ethnicity and teaching modalities; gaining insights into the factors contributing to performance gap(s) of these two assessment approaches; and proposing recommendations for improvement.

2 Background

Programming skills are an essential component of computer science education. Programming skills can be improved and monitored with weekly programming assignments and/or programming projects [1]. In addition, programming exams have been shown to facilitate the learning of programming skills [2, 3]. Programming exams have been used as an assessment method of learning and teaching [4, 6, 7]. Integration of a programming exam as an assessment method ensures that students do not share their code with each other, that students can concentrate on solving problems and logic of implementation rather than fixing bugs that can be caught by a compiler, and computer-based programming exams have the same familiar programming environment that students use to complete their programming assignments.

Consequently, a programming exam was incorporated in CS3 at CSU Chico since Fall 2018 to give students adequate opportunity to demonstrate their programming skills and to motivate them to better engage in outside-class and in-class programming practice. In addition, students were offered opportunities to retake poorly completed written tests after receiving tutoring from the instructor. This research focuses on analyzing whether programming exams and Tutor-Retake written tests affected performance gaps (if any) between student groups by gender, ethnicity, and teaching modalities.

3 Methodology

CS3 is the third programming course in a three-course sequence taught in C++. Students taking CS3 have programming experience in basic C++ constructs, C++ classes and pointers, and fundamental data structures such as an array, list, queue, stack, and binary search tree. In CS3, the key topics covered are time and space analysis, a balanced search tree, and graphs. Three majors — Computer Science, Computer Information Systems and Computer Engineering — require all three programming courses.

In this retrospective study, data were used from only these three majors to reduce potential bias resulting from the motivation to take a required versus a not-required course. Here, student performance and demographic data for CS3 were included from eight semesters: (1) the three semesters taught in person before the COVID pandemic (Fall 2018, Spring 2019, and Fall 2019); (2) the first half of Spring 2020 taught in person and the second online; (3) the following three semesters, all taught online (Fall 2020 and Spring 2021 due to COVID pandemic and Fall 2021 due to the instructor's medical issues); and (4) Fall 2022, taught in person following the COVID pandemic campus response.

The pre-existing performance information was collected via Blackboard, an LMS, as a normal educational practice, and consisted of (1) an average score on written tests (not including the final exam); (2) the score of the final exam; (3) a flag indicating that at least one problem on all programming exams was solved and implemented correctly; (4) a flag indicating that all problems on all programming exams were completed correctly; and (5) the overall grade in the course. The demographic data used for this paper was collected as normal business practice by the university and included (1) age; (2) major; (3) gender; and (4) ethnicity. Data for students in non-required majors and for students younger than 18 years old were removed, leaving data for 407 students in the overall dataset.

To answer the research questions, a Chi-square with a Yates corrections test was used, together with a contingency table in which rows represented groups (gender/ethnicity/teaching modality) and columns corresponded to the five performance categories. Satisfactory outcome for the average score on all written tests and the final exam's score was defined as scores at or above 70%; and consequently, unsatisfactory outcomes corresponded to scores below 70%. Satisfactory outcome on overall grade was a score equal to or greater than 73% (the passing score for the course).

To evaluate students' performance in Programming Exams, the count of students who correctly completed at least one problem in all programming exams and the count of students who finished all problems in all programming exams were used. The Spring semester of 2020 had one programming exam, and the remaining seven semesters had two programming exams. Four semesters had three written tests and the other four semesters had two written tests covering the same material as the three written tests. The characteristics of written and programming exams are provided below.

For analyses involving gender, the total number of students was 407 (all students available) with 367 (90.2%) male and 40 (9.8%) female students. The analysis by ethnicity excluded students whose ethnicity was unknown and students marked with non-resident alien status. The total number of students in this analysis was 366. White, Asian and the students with two or more ethnicities (non-Hispanic) were considered Ethnicity Majority, with a total count of 279 (76.2%). The remaining students (Hispanic, African American and Native Hawaiian) were categorized as Ethnicity Minority with a count of 87 (23.8%). To study students' performance data in relation to teaching modality (in-person versus online), the data from Spring 2020 was excluded, since it was split into two phases by the COVID campus shutdown and used two different teaching modalities. The total number of students across the eight semesters who were taught in person was 209(59.4%), and the number online 143 (40.6%), with a total of 352 used in the analyses.

All analyses were performed on the data both with outliers and without outliers. An outlier in an average score on written tests category was defined as a score of 0 on the second or third written test (including students who did not take one of the tests). For the remaining performance categories an outlier was a final exam score of 0 (students who did not take the final exam).

3.1 Programming Exam Characteristics

The number of students in CS3 was 50-60 per semester. When a course was taught in person, a programming exam was conducted during lab time for 1 hour and 50 minutes in the room with 30 computers, the same room where students had their labs. There were two sessions of lab time on two different days, each with up to 30 students. For a single session there were two different versions of a programming exam, printed copies of which were distributed alternatively among students so that no two students sitting next to each other had the same version. The second session had different two versions from the first session. The students have not seen exact problems before the exam. It was an open book exam: each student could use any code developed by himself in this course and any lecture notes. Each programming exam was an individual assignment, and students were not allowed to ask for help from the instructor or a lab assistant. Computers were networked and had the access to internet. During the exam time, the only website that was allowed to be kept open was the university's website for submission of programming solutions. For online teaching, there was a single version of a programming exam and students had about 2-3 days to complete it. Students were not allowed to look for solutions online or ask for help from anyone.

The content of both programming exams reflected the course's learning outcomes. The first programming exam was on the AVL-tree and the second on graphs. For the first exam, students had to solve two problems and implement their solutions. Some code developed previously in the course was provided, namely, a declaration and definition of an AVL-tree class with basic functions maintaining balance of a tree. Students had to write member functions for the AVL-tree class with solutions for the required problems, compile and debug their programs using the provided tests files with different test cases, each containing an input and expected output files. The first problem was on tree traversal: students were required to use a specific tree traversal (pre-order, in-order or post-order) to print out specific data stored at each node of a tree. The second problem was more difficult: students had to write a function(s) whose running time is in O(log n). For each problem, the instructor provided definitions of the involved concepts, description of a problem to be solved, and an example with a figure of an AVL-tree illustrating the problem and its solution. The second programming exam had only one problem for most of the semesters, and two or three problems for Fall 2022. Students were provided with the code containing a declaration and definition of a class Graph together with some member functions including *bfs* (breadth first search) and *dfs* (depth first search) functions. The programming exam's instructions included relevant definitions, problem description and an example with a graph figure and explanation of a solution for this graph. In addition, for the second programming exam, an outline of a solution for the problem was provided in English, and students had a choice either to implement the provided solution or to come up with their own solution. Time complexity was not restricted. For Fall 2022, the second programming exam contained two or three problems with one problem similar to the rest of the semesters, and the other problem(s) easier. The content of programming exams during online teaching was like the content during in-person teaching before the COVID pandemic.

The grading policy was carefully explained before the programming exam. Each programming exam was worth 5% of the final grade. Students were provided with the files containing different test cases: input and expected output for each problem. Students were required to write a program that would pass all test cases for a problem to receive credit for that problem.

Prior to taking a programming exam, students had completed at least two programming homework assignments on a relevant topic and practiced solving similar problems during at least two lectures with the solutions explained by the instructor.

3.2 Written Tests characteristics

There were two written tests for half of the semesters and three for the other half. Each test was individual work. All tests taken in person were open books where students could use their own lecture notes or any printed notes provided via the university's LMS. Time per test was 75 minutes. The room held all students in class. The content covered by two tests was similar to the content covered by three tests. Each test consisted of 8-12 problems related to learning outcomes of the course. No multiple-choice problems were given. The problems on a written test were similar to the ones discussed in class, but never identical.

Grading requirements for a written test in CS3 taught in person before COVID pandemic were the following: in the first semester (Fall 2018), students were required to receive at least 70% on the average over two tests (not including the final exam); and in the remaining two semesters (Spring 2019 and Fall 2019), students were required to receive at least 70% on each test. If a student did not meet this requirement, they did not receive a passing grade for the course. To support students in meeting this stringent requirement, two additional attempts were permitted as long as the student came for tutoring with the instructor. The grade received on each test was the best grade received in all attempts for that test. In Fall 2022, when teaching in person after COVID pandemic, there was no special passing requirement based on a score of a written test; hence, students were permitted only one attempt on each test. During teaching online, the students were given a take-home test and had at least 8 hours to complete it. Students were not allowed to look for solutions online or ask anyone for help.

In-person students had 1 hour 50 minutes to complete the final exam, and online students had at least 6 hours. The final exam was individual work for both teaching modalities, and consequently students were not allowed to ask for help. Each student had only a single attempt to finish the final exam.

3.3 Effectiveness of Tutor-Retake Written Tests

To study the effectiveness of allowing students to retake a written test after tutoring with an instructor, performance categories were analyzed with two groups of students: students taking CS3 in person (in a classroom) before the COVID pandemic and students taking CS3 in person after the COVID pandemic. The total number of students was 209, with 159 (76.1%) in the first group and 50 (23.9%) in the second group. The first group of students was given an option to take a Tutor-Retake written test with up to three attempts, and the third group was allowed only a single attempt on each test.

4 Results and Discussion

The results of the analysis of the entire data set are shown in Table 1. All three tables show results in one-tailed p-values. Statistically significant p-values are shown in bold font. Performance categories "One on PE" and "All on PE" stand for "At least one problem was solved and implemented correctly on all programming exams" and "All problems were solved and implemented correctly on all programming exams" respectively.

According to these results, there are no significant performance gaps by gender, except for the performance category labeled as "All on PE" in Table 1. In fact, there is a strong statistically significant association for this performance category in all groups of data used in this research (by gender, ethnicity and teaching modality).

Table 1: One-Tailed P Value for Chi-Square Analysis of Entire Data Set

Performance Category	Gender		Ethnicity		Teaching Modality	
	With Outliers	No Outliers	With Outliers	No Outliers	With Outliers	No Outliers
Written Tests	0.22	0.18	**0.01**	**0.03**	**0.0001**	**0.0001**
Final Exam	0.08	0.08	0.11	0.31	**0.001**	**0.004**
One on PE	0.12	0.1	**0.001**	0.11	0.14	**0.04**
All on PE	**0.02**	**0.02**	**0.0003**	**0.002**	**0.007**	**0.002**
Overall Grade	0.19	0.08	0.11	0.47	**0.009**	**0.045**

Interestingly, for analysis by teaching modality, the association between rows and columns of corresponding contingency tables is statistically significant for all performance categories (with no outlier cases). Students taking CS3 online had access to video-recorded lectures, had more time to complete a written test, the final exam and a programming exam, and had the opportunity to ask for help and/or search online for solutions. Nonetheless, it seems that this group of students performed less well than the students taking CS3 in person. Even though the instructor kept students engaged during Zoom lectures by asking students questions (students were given points for this participation) and by requesting that students write training exercises and submit their notes electronically for points after lectures, online students still did not do as well as the students in on-site classes. Many different factors could contribute to this overall weaker performance: (1) it is more challenging to maintain attention online, or (2) it is more appealing to miss a lecture with plans to watch it later via a recording but not following through.

When analyzing the entire data set and grouping students by ethnicity, there was a statistically significant difference in performance on Written Tests. To investigate this further, the same analysis was performed for the group of students taking CS3 in person before COVID. This group of students received extra support on written tests in the form of tutoring and multiple attempts for each test. Table 2 shows that this support was effective since there was no statistically significant association between groups of students by ethnicity and average score on all written tests.

Table 2: Performance Gaps by Ethnicity in the Dataset In-Person Tutor-Retake Written Tests

Performance Category	Ethnicity	
	With Outliers	No Outliers
Written Tests	0.15	0.3
Final Exam	0.15	0.43
One on PE	**0.02**	0.26
All on PE	**0.0073**	**0.02**
Overall Grade	0.24	0.4

While comparing two groups of students, both having taken CS3 in person but having different requirements and different support for written tests (Table 3), the statistically significant difference is observed in performance on the written tests category, implying that the Tutor-Retake written test is more effective. Surprisingly, there was no difference in performance on the final exam between these two groups. The final exam was a cumulative exam with problems similar to those on written tests. It was expected that the students who received extra support (tutoring and additional attempts on each test) would have performed better than those who did not get the support. The analysis showed otherwise.

Table 3: Effect of Tutor-Retake Written Tests on Performance In-Person Before and In-Person After COVID Pandemic

Performance Category	With Outliers	No Outliers
Written Tests	**0.005**	**0.001**
Final Exam	0.27	0.29
One on PE	0.19	0.11
All on PE	**0.011**	**0.001**
Overall Grade	0.35	0.39

One explanation might be that by the time students got to the final exam, they had sufficient training and practice, and had completed a review for the final exam using similar problems, so that this was enough for them to develop necessary confidence and skills to perform well on the final exam. Alternatively, this also might indicate that the final exam was not sufficiently difficult to distinguish between those students who grasped the concepts at depth, and the rest of the students.

There was no difference in performance measured by overall grade, except in the case of online instruction. This might be explained by the weighting of the overall grade. Students who regularly attended lectures and labs were better prepared to do well on homework assignments. Moreover, participation during lectures and labs contributed to the overall grade. In addition, students had videos that outlined solutions and implementation approaches for many homework problems, which helped them with the homework assignments. The requirement to do at least 70% of the work on each project also likely motivated students to do well on projects that also reflected in the overall grade.

5 Conclusions

The performance category that indicated that students had solved all problems on all programming exams identified performance gaps in all groups. In [7], the authors encountered the same situation on one kind of problem for programming exams, and they concluded that their ambitions were too high, and that the problem should be changed. This is one possibility – to change problems that present difficult challenges for most students. Another option is to train students to solve problems that involve synthesis skills starting at CS1 and CS2 and continuing in CS3 (this follows from the fact that the most challenging problem on all programming exams required synthesis skills).

The present study showed the effectiveness of a Tutor-Retake written test. Unfortunately, this technique, although very helpful for students, is very time consuming for instructors, which calls for future development of sophisticated automated test preparedness tools individualizing the tutoring process for each student. In general, programming exams are a very effective technique that allows students to demonstrate their skills in a familiar environment and motivates them to further develop programming skills. This study showed that the majority of students managed to solve and implement at least one programming problem in a two-hour period. In addition, it demonstrated that overly-ambitious problems requiring higher levels of thinking probably are not the best way to assess students during a programming exam.

References

[1] Joe Michael Allen et al. "An analysis of using many small programs in cs1". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019, pp. 585–591.

[2] João Paulo Barros et al. "Using lab exams to ensure programming practice in an introductory programming course". In: *ACM SIGCSE Bulletin* 35.3 (2003), pp. 16–20.

[3] Kevin Buffardi and Subhed Chavan. "Is programming relevant to CS1 students' interests?" In: *Journal of Computing Sciences in Colleges* 37.1 (2021), pp. 45–53.

[4] Quintin Cutts et al. "Laboratory exams in first programming courses". In: *Proceedings of 7th Annual Conference of the Higher Education Academy for Information and Computer Sciences*. 2006, pp. 224–229.

[5] Sabine Hammer, Martin Hobelsberger, and Georg Braun. "Using laboratory examination to assess computer programming competences: Questionnaire-based evaluation of the impact on students". In: *2018 IEEE Global Engineering Education Conference (EDUCON)*. IEEE. 2018, pp. 355–363.

[6] Norman Jacobson. "Using on-computer exams to ensure beginning students' programming competency". In: *ACM SIGCSE Bulletin* 32.4 (2000), pp. 53–56.

[7] George Stockman et al. "CS1 and CS2 Programming Exams for Assessing Learning and Teaching". In: *2004 Annual Conference*. 2004, pp. 9–358.

A Heap Data Structure for Weighted Random Selection with Dynamic Weights[*]

John A. Stratton, Andrew Kuhlken, and Zechariah Frierson
Computer Science
Whitman College
Walla Walla, WA 99362
{strattja,kuhlkena,friersoz}@whitman.edu

Abstract

This paper presents a new heap-based data structure for weighted random selection with dynamic updates. Our method delivers $O(\lg N)$ selection and weight update in the worst case, but also leverages the heap property to reduce the average depth of the elements most likely to be selected, improving performance for heavily skewed distributions. This method is asymptotically equivalent to and experimentally competitive with widely used standard library implementations based on a binary search algorithm for static distributions, while also enabling efficient dynamic weight updates.

1 Introduction and Background

Many applications rely heavily on the fundamental operation of randomly selecting an element from a set given an arbitrary discrete probability distribution. This has motivated the development of many algorithms and data structures for optimizing weighted random selection (WRS) [1] and its inclusion in the standard libraries of programming languages. However, most prior work assumes WRS from a *static* distribution, incurring linear or super-linear costs any time weights are updated. Important applications, like some discrete event simulations, require both efficient selection and efficient individual

[*]Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

weight update operations, which we will refer to as *dynamic* weighted random selection (DWRS). For instance, to perform a stochastic chemical simulation using the classic Gillespie algorithm, a simulation algorithm iteratively calculates "propensities" for each reaction, using them to randomly choose which reaction will occur next, and updating the state of the system and reaction propensity values [5]. When scaling up to multicelluar systems, it is common for simulations to involve thousands of reactions but only require a constant number of propensity updates on each event. The lack of an efficient, general algorithm for DWRS motivated the development of ad-hoc optimizations that were practically effective for specific use cases, despite poor worst-case performance in general [2], and the development of alternative simulation algorithms to avoid WRS entirely [4].

We formally define DWRS as an abstract data type representing a indexed collection of nonnegative weights w_j, with ordinals $j \in [0 : N)$, supporting operations `select()` and `set_weight(i,w)`. The `select()` operation must return an element at random, returning element i with probability $\frac{w_i}{w_S}$ where $w_S = \sum_j w_j$. The `set_weight(i,w)` operation sets the weight of element i to a new value. We describe a new data structure implementing each of these operations in $O(\lg N)$ time in general, which approach $O(1)$ selection as the weight distribution becomes more skewed.

We will build our contributions in two sections. Section 2 introduces a simple complete tree data structure implementing the key DWRS operations in $O(\lg N)$ time. Section 3 extends that concept with a max heap, explaining how this optimization provides practically relevant performance improvements. In Section 4, we analytically compare our algorithm to other published work, identifying how our work captures benefits for relevant use cases without compromising generality. Section 5 presents the methodology and results of empirically evaluating an implementation of our method in C++. We summarize our contributions in Section 6.

2 Random Selection with a Weightsum Tree

We define a weightsum tree as a complete binary tree, where every node x stores the weight of the element at that node w_x, as well as a subtree weight sum for the subtree rooted at that node, sum_x. Given that definition, we can define algorithms for selection and weight update by Algorithms 1 and 2.

Algorithm 1 is a recursive procedure selecting a node from a tree rooted at x given a random number in the range $[0, sum_x]$. If the random number is smaller than the weight of the root node, that node is selected. This will occur with probability w_x/sum_x as desired. If the random number is larger, then the algorithm must instead select a node from one of the two sub-

```
Procedure select()
    Generate uniform random variable $r$ from the range $[0, sum_{root}]$;
    return selectFromSubtree($root$, $r$);
Procedure selectFromSubtree($x$, $r$)
    if $w_x > r$ then
        return $x$
    end
    if $r < w_x + sum_{x.left}$ then
        return selectFromSubtree($x.left$, $r - w_x$);
    else
        return selectFromSubtree($x.right$, $r - (w_x + sum_{x.left})$);
    end
```

Algorithm 1: Weighted Selection

```
Procedure setWeight($x$, $v$)
    $w_\Delta \leftarrow v - w_x$;
    $w_x \leftarrow v$;
    $sum_x \leftarrow sum_x + v$;
    while $x$ is not root do
        $x \leftarrow x.parent$;
        $sum_x \leftarrow sum_x + w_\Delta$;
    end
```

Algorithm 2: Set weight of item x to the nonnegative value v

trees. In this case, the appropriate probability of selecting a node from the left subtree should be $sum_{x.left}/(sum_{x.left} + sum_{x.right})$. Because $sum_x = sum_{x.left} + sum_{x.right} + w_x$, the algorithm makes this selection by subtracting w_x from the original random number, yielding a uniform random number between $[0, sum_{x.left} + sum_{x.right}]$ and choosing to traverse left if it is less than $sum_{x.left}$. In the case that the algorithm traverses right, it also subtracts $sum_{x.left}$ from its random argument, such that in either recursive case, the passed random variable argument is a uniform random variable in the range $[0, sum_x]$ for the new root. We can therefore inductively conclude that the algorithm must have the appropriate likelihood of selecting each node in the tree.

Algorithm 2 traverses up the tree from the node whose weight is being changed to update the subtree weights of that node and all its ancestors. Since the height of a complete tree is $\Theta(\lg N)$, selection and update according to these algorithms are both $O(\lg N)$ operations. A practical implementation will require several other supporting algorithms, such as initializing the data structure from a set of weights, but such methods are straightforwardly implemented based on the algorithms already described.

3 Heap-ordered Weightsum Tree Optimization

While the simple weightsum tree approach yields asymptotically efficient worst-case selection and update operations, for some probability distributions it is possible to further improve the average execution time. In particular, the amount of work done in selection is proportional to the depth of the selected element in the tree. For uniform probability distributions, each node is equally likely to be selected, so there is no benefit to reordering nodes. However, for heavily skewed probability distributions where a small number of nodes represent a disproportionate amount of the weight of the elements in the tree, the placement of those particular nodes in the tree can significantly affect the performance of the selection operation. To minimize the expected depth of the nodes of greatest weight in the collection, we can extend the weightsum tree by adding a max-heap ordering constraint that sorts the nodes with the greatest weight towards the top of the tree. Supporting this feature requires two meaningful changes to our prior procedures. First, because elements can move around in the tree, each node must now also store the element index of the element currently being held in that position. For efficient weight modification based on element index, we must also maintain a separate table storing for each element index the tree position that currently holds that element. With these changes, our selection procedure is practically unmodified, except that we return the stored element index of the selected node rather than the node position itself. Our weight update procedure, however, must be modified to preserve the max-heap property, as shown in Algorithm 4.

In the worst case, these changes make no difference asymptotically or practically. In fact, the overhead of the additional storage and additional work being performed to order the heap may degrade performance for some applications. However, these relatively small losses in many situations can enable significant benefits in particular situations that have motivated entirely unique methods that can be subsumed by this data structure.

One important situation is dealing with highly skewed data distributions. Because of the heap ordering, the element most likely to be selected is by definition at the root of the tree and considered first for selection. Even if a small subset of nodes represent the majority of the weight, those nodes will collect towards the top of the tree at small depths, making the 87expected selection time approach $O(1)$ as the probability distribution tends towards a constant number of nodes holding practically all of the collection's weight.

Another important consideration is near-static data sets. While truly static

```
Procedure swapNodes(c, p)
    $w_\delta = w_c - w_p$;
    $sum_c \leftarrow sum_c - w_\delta$;
    $sum_p \leftarrow sum_p + w_\delta$;
    swap($element_c, element_p$);
    swap($w_c, w_p$);
```
Algorithm 3: Swap the elements and weights of a parent and child node.

```
Procedure setWeightHeap(a, v)
    $x = PositionOf[a]$;
    $w_\triangle \leftarrow v - w_x$;
    setWeight(x, v);
    if $w_\triangle > 0$ then
        while x is not root AND $w_x > w_{x.parent}$ do
            swapNodes(x, x.parent);
            $x \leftarrow x.parent$;
        end
    else
        while x is not leaf AND $w_x < max(w_{x.left}, w_{x.right})$ do
            $c \leftarrow$ larger weighted child of x;
            swapNodes(c, x);
            $x \leftarrow c$;
        end
    end
```
Algorithm 4: Set weight of item x the to nonnegative value v

data sets can take advantage of high up-front data structure construction costs for many repeated selection operations, there are many applications where distributions are not static, but the magnitude of updates is typically very small. The cost of updating the weight of a node in the heap is proportional to the sum of the node's depth, which determines how many ancestors need their cumulative weight sums updated, and the distance the node has to move in the heap given its new value. In a near-static data set, weight updates are small and unlikely to require moving the node position at all, so the weight update cost is dominated by the depth of the node yet again. This means that the heap data structure performs best when the elements being updated frequently are also the weights most likely to be selected, which can be the case in some applications, particularly when a weight only needs updating after selection.

4 Comparative Asymptotic Analysis

Several other methods have been proposed for weighted random selection, including methods for dynamic weighted random selection. However, many of them are optimized for specific use cases such that a generalized mixture of selection and update operations on an arbitrary distribution are worse than $O(\lg N)$ per operation.

Linear search can be performed with only an additional sum variable by simply iterating over the weights until the sum of traversed elements exceeds a randomly generated number in the range $[0, sum]$. This algorithm is effectively searching for the first element where the cumulative sum of that element and all preceding elements is larger than the random selector variable. Prior work has proposed sorting elements to increase the efficiency of linear search to $O(1)$ in the case of skewed data distributions [2]. **Binary search** is a method often implemented in standard libraries and computes an array of cumulative sums at construction. This array of cumulative sums is a sorted array on which a classic binary search algorithm can be performed to find the smallest element larger than the generated random selector variable. This method is very similar to ours in terms of selection but requires $O(N)$ updates to the cumulative sum array when a single element changes. The fastest methods for random selection are **Alias Selection** methods which can achieve $O(1)$ selection at the expense of $O(N)$ data structure construction that does not support efficient incremental updates [7]. For data distributions that are near-static, there are extensions to provide expected $O(1)$ selection and update times using a rejection method, but those methods still require $O(N)$ data structure maintenance any time elements change too much [6]. Other closely related work includes Fenwick trees [3], which is another tree structure where nodes store subset sums of weights and could likely be extended for application to the DRWS problem, but the properties of heaps are more thoroughly characterized by preexisting literature.

5 Experimental Evaluation

We implemented the weightsum tree and heap-ordered weightsum tree data structures in C++, and tested their performance as well as the discrete distribution class in the standard template library. Each test involved generating a set of weights from a random distribution and then measuring the time to complete a set of queries. We varied the distributions from which the weights were drawn to examine how sensitive different implementations were to variations in weight distribution. Because our weights were drawn randomly, each of our results is presented as the average across 100 trials to control for ran-

	Selection	Update	Notes
Linear search	$O(N)$	$O(1)$	$O(1)$ selection for skewed and sorted distribution
Binary search	$O(\lg N)$	$O(N)$	
Weightsum tree	$O(\lg N)$	$O(\lg N)$	
Heap-ordered Weightsum tree	$O(\lg N)$	$O(\lg N)$	$O(1)$ selection for skewed distribution, $O(1)$ update for skewed and near-static distribution.
Alias selection	$O(1)$	$O(N)$	
Rejection method	$O(1)$	$O(N)$	$O(1)$ near-static update

Table 1: Asymptotic comparisons of dynamic weighted random selection methods.

Uniform	Random integers generated uniformly in the range $[1 - 10]$
Normal	Random numbers generated from a normal distribution with standard deviation 2, mean shifted dynamically such that all generated numbers are non-negative.
Weibull	A highly skewed distribution of random numbers drawn from a Weibull distribution with parameter 0.5

Table 2: Descriptions of weight distributions in test cases.

dom variations in the weight distributions. For static tests, each query was simply a selection operation, accumulating the selected indexes into a sum to ensure the compiler did not remove the selection in dead code elimination. Dynamic tests involved running an equal number of iterations with each iteration performing one selection operation and one weight update operation on the selected element, giving it a new weight drawn from the same original probability distribution.

All applications were compiled by gcc 11.3.0 with the -O3 flag and executed on an Intel Xeon Bronze 3204, recording the time to execute 10^6 iterations of operations. Recorded time was measured with the gettimeofday() function within the test application. Weights were generated for different experiments according to the distributions described in Table 2. Software used for these experiments can be accessed at the URL below.
`https://github.com/WhitmanOptiLab/discrete_distribution`

Figures 1 and 2 show the results of our experimental tests. The log-scale axis reveals the expected logarithmic scaling of all tested methods for the different distributions included as nearly linear runtime growth. We can also see

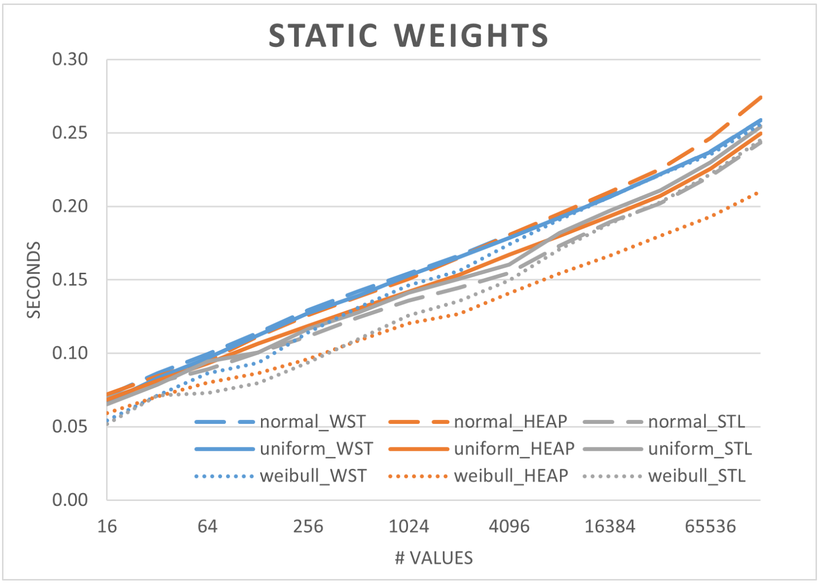


Figure 1: Performance of selection on a static probability distribution.

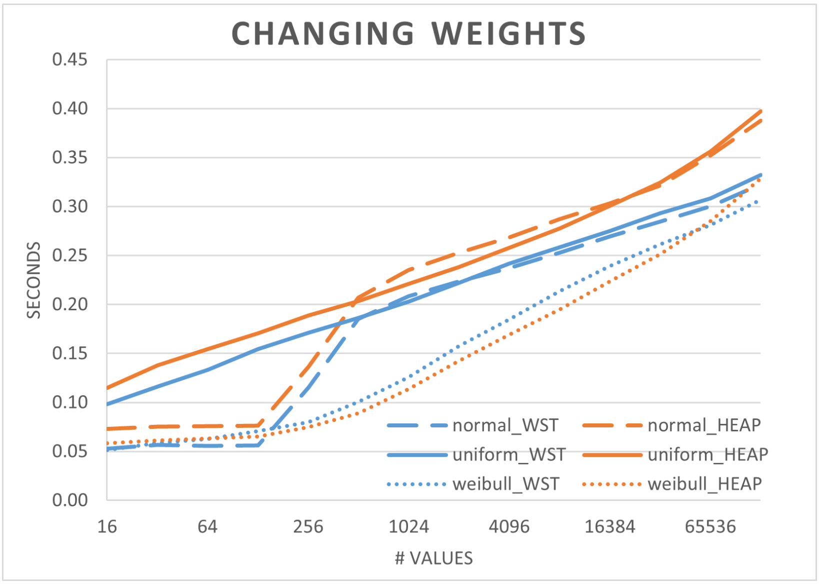


Figure 2: Performance of selection and update on a data distribution.

that for these static distribution methods, all methods perform better on the skewed distribution, likely because predictability in the path to the selected item improves hardware system performance in general. However, only the heap data structure maintained that performance benefit as the dataset size increased, because the heap ordering kept the elements with the highest weight easily accessible no matter how big the overall data collection got. We can also see that as the data sizes increase, the heap data structure performance varied much more than the performance of either of the other data structures with the weight distribution, as expected. Our weightsum tree was generally slightly slower that the binary search method implemented in the standard library.

We did not test the standard library for changing weights, as each modification would have required reconstructing the object with a linear-time overhead and clearly underperformed our implementations. When weights are allowed to change, both the Weibull and normal tests performed better for small dataset sizes. This is because when the highest weighted elements get selected and are reassigned a new value from the distribution, the collection of elements will skew towards the lower end of the distribution of values. Out of 10^6 selections, it is very likely that most elements are eventually assigned extremely low probabilities, effectively skewing the dataset more and more over time. Once the dataset size grew enough that our fixed number of updates had less effect on the overall distribution, we can see that only the Weibull distribution maintained favorable performance over a wider range of sizes. In general, the heap distribution did impose some additional overhead compared to the basic weightsum tree when the heap ordering had less beneficial impact, but it outperformed in the skewed Weibull tests over the majority of the sizes. As the sizes reached the upper end of our test range, we saw across datasets that the heap data structure started to experience caching issues before the other methods due to its increased memory footprint.

6 Conclusions

Dynamic weighted random selection has been important to a variety of applications, but currently available solutions all lack efficiency across a broad range of use cases. The weightsum tree, with or without heap ordering, is not best-in-class for most use cases. Static or near-static distributions can be supported with faster selection algorithms. Highly uniform or highly skewed distributions could each be tailored to more efficient methods as well. However, our proposed weightsum tree has no catastrophic worst-case performance cases, and can, in at least some situations, capture meaningful performance improvements from those specialized cases.

References

[1] Karl Bringmann and Konstantinos Panagiotou. “Efficient Sampling Methods for Discrete Distributions”. In: *Algorithmica* 79.2 (Oct. 2017), pp. 484–508.

[2] Yang Cao, Hong Li, and Linda Petzold. “Efficient formulation of the stochastic simulation algorithm for chemically reacting systems”. In: *The Journal of Chemical Physics* 121.9 (2004), pp. 4059–4067.

[3] Peter M. Fenwick. “A new data structure for cumulative frequency tables”. In: *Software: Practice and Experience* 24.3 (1994), pp. 327–336.

[4] Michael A. Gibson and Jehoshua Bruck. “Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels”. In: *The Journal of Physical Chemistry A* 104.9 (2000), pp. 1876–1889.

[5] Daniel T. Gillespie. “Exact stochastic simulation of coupled chemical reactions”. In: *The Journal of Physical Chemistry* 81.25 (Dec. 1977), pp. 2340–2361.

[6] Vo Hong Thanh, Roberto Zunino, and Corrado Priami. “Efficient Constant-Time Complexity Algorithm for Stochastic Simulation of Large Reaction Networks”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 14.3 (2017), pp. 657–667.

[7] Alastair J Walker. “An efficient method for generating discrete random variables with general distributions”. In: *ACM Transactions on Mathematical Software (TOMS)* 3.3 (1977), pp. 253–256.

Choosing Our Computing Birthplace: VSCode vs Colab as GenEd IDEs[*]

Elena Miller, Katy Shaw, and Zachary Dodds
Computer Science Department
Harvey Mudd College
Claremont, CA 91711
{elmiller, kshaw, dodds}@g.hmc.edu

Abstract

First impressions are important. The initial environment in which our computing students express themselves helps shape their foundational understanding of *what computing is*, *what it's for*, and *who participates.* This work distills experiences and insights from offering Comp1 and Comp2[1] with two different IDEs: Microsoft's VSCode and Google's Colab. We identify and describe several axes along which we compare our students' experience of these two. This effort has changed the way we offer Comp1, a degree requirement of all students at our institution, and Comp2, an optional follow-up course, required by some computationally-themed programs.

1 Choosing our hometown?

Our birthplace exerts a powerful and lifelong force[8]. Although it does not grow roots as deep as our first language, culture, and family, our *computing birthplace* – the initial environment in which we express ourselves *executably* –

[*]Copyright ©2023 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

[1]In this work, Comp1 and Comp2 are generic CS1/CS2 names. At our institution, Comp1 is a GenEd degree requirement of every student. Comp2 is a choice, required by some programs.

has outsized impact on our students' relationship with computing. As a formative experience, it echoes long after the syntax - and instructor - of "Comp1" have faded.

Even for computing identities born elsewhere, we choose a present-day environment for our students to share: a *computational hometown*, perhaps. That environment sets the stage for shared experiences, community-building, and students' computational identity and self-efficacy.

In light of Comp1's evolution toward the role of universal GenEd at more and more institutions and programs, this work asks, "How do we decide on a hometown - or birthplace - for our students?"

2 VSCode and Colab

Let's acknowledge: the topic of computing environments is fraught! This truth shows the importance of our computational birthplaces. We may explore - even settle - in other realms, but we are enduringly shaped by our first.

What's more, there are many factors that influence the choice of computational birthplace (or hometown). Not all these factors are under our control. Indeed, it's worth asserting that the natural analogy always holds: *We don't have to decide!* Sometimes, holding on as the world spins is all the agency we have, and we are right to allow ambient forces to decide for us.

Other times, we have more agency. As a part of a consortium of small schools, we found the opportunity to run our introductory-computing courses, Comp1 and Comp2, in two different ecosystems: Microsoft's Visual Studio Code (VSCode)[6] and Google's Colaboratory (Colab)[3]. Briefly, VSCode is a widely-used professional editor/integrated development environment (IDE); Colab is a widely-used notebook interface supporting interlaced context and code cells. The next section shares the axes along which we have compared these two (and other) IDEs for introductory college computing, especially when a GenEd required of all students.

Our conclusion is not unexpected: *There are only good birthplaces.*

Yet we find there are also real differences in the *specific strengths and drawbacks* a particular IDE conveys as an initial experience. In sharing our (ongoing) journey, we hope these axes can help other schools and instructors as, together, we invite all students into their computational futures.

3 Our Opportunity Map

Figure 1 shows a large number of code-editors and IDEs across two important axes: the size of the community of current users and the ubiquity/suitability for professional settings. There is more than a little subjective judgment along both axes. Yet we found these comparisons helpful in conceptualizing the landscape of possibilities.

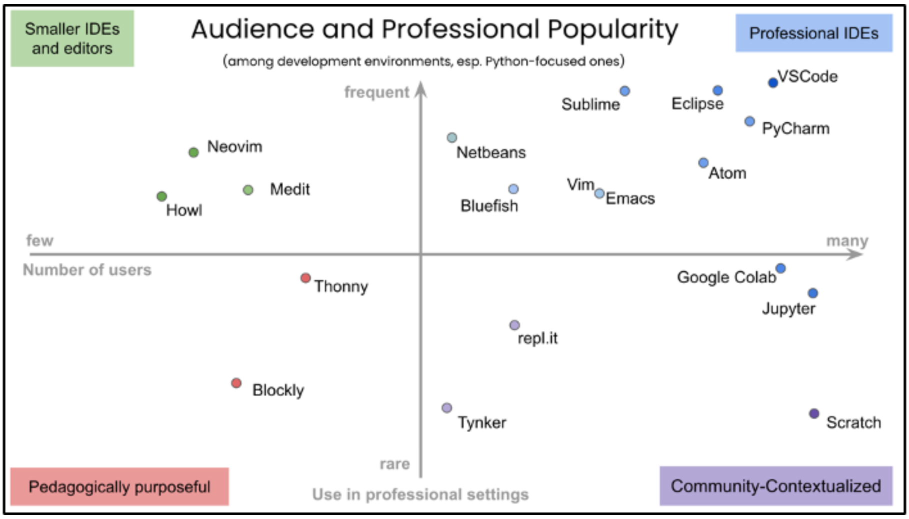
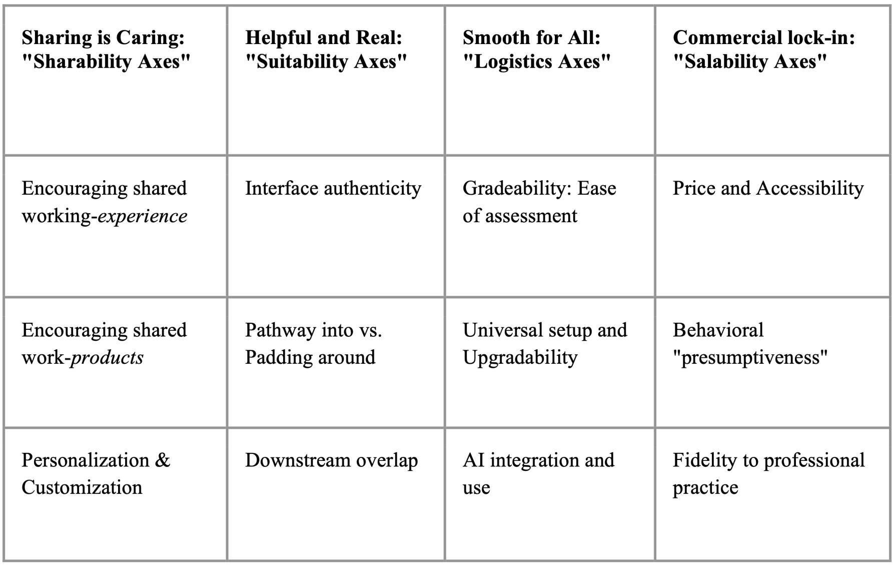


Figure 1: A comparison of several editors and IDEs (i.e., computational birthplaces), especially ones that might be used for Python/GenEd, along axes representing the relative size of the user base (horizontal) and ubiquity/suitability for professional practice (vertical). There is more than a little subjective judgment here: this provides a conceptual map of a “zeroth-order" landscape of possibilities.(Seasoned cs’ers and software engineers will appreciate that vim and emacs share a single point.)

For us, the important features of Figure 1 were (a) that both VSCode and Colab offer large communities of fellow users, and (b) that they differ in fundamental purpose. VSCode intends to serve software engineers well. It does. Colab intends to contextualize and share computational exploration. It does.

Because both environments are so successful, and because there are so many considerations that comprise a large, important, and widely shared college course, we expanded the dimensionality of our space. Figure 2 provides the twelve criteria, or “axes," along which we have contrasted VSCode and Colab. They are grouped into four rubrics: (1) sharability, important in establishing a campus-wide baseline for computing work, (2) suitability, that is, suitability for college students new to computing, (3) “sanity-preservation," which assesses logistical support for instructors and instructional realities, and (4) salability, i.e., how internal and external audiences feel about - and make themselves felt through - the interfaces.



Sharing is Caring: "Sharability Axes"	**Helpful and Real: "Suitability Axes"**	**Smooth for All: "Logistics Axes"**	**Commercial lock-in: "Salability Axes"**
Encouraging shared working-*experience*	Interface authenticity	Gradeability: Ease of assessment	Price and Accessibility
Encouraging shared work-*products*	Pathway into vs. Padding around	Universal setup and Upgradability	Behavioral "presumptiveness"
Personalization & Customization	Downstream overlap	AI integration and use	Fidelity to professional practice

Figure 2: An expanded set of axes by which we have compared the characteristics of VSCode and Colab as introductory computing environments for all students. The twelve considerations naturally group into categories of *sharability*, *Comp1- suitability*, *logistics support (instructors and students)*, and *"salability"* for each.

The next section elaborates on our - and our students' - experiences along each of these considerations.

4 Axes to Grind

4.1 Sharing is Caring

Introductory courses provide experiences shared by an entire class – in our case, an entire class-year. This common experience is important for creating connections and strengthens the sense of community within the class. Of course, *any* commonly used environment serves the purpose of **enabling shared work in subsequent courses**. For us, Colab and VSCode both have strong downstream proponents, with computational courses leaning toward VSCode and applied-science courses toward Colab notebooks. Even if there is a bigger-picture shift between years of changing code editors, the current class still shares their introduction to CS experience with all of their peers in their class. We find this an enormous advantage. *On this axis:* **Toss Up**

Sharing is caring and **communication of ideas and work products** opens the door to collaborations and community-building. As a Google document, Colabs have strong sharing features: multiple people can see and access a Colab at one time, though only one can edit at a time. As a local editor, VSCode does not support this capability out-of-the-box. However, its substantial library of extensions offers Live Share[4], a capable shared-editing experience – one we use extensively in CS3, as it requires additional set up/configuration. For our introductory Comp1/Comp2 experiences, the ease of Colab tops VSCode's deep bench. *On this axis:* **Colab**

The chance to actively **personalize/customize** one's environment goes a long way to establishing a sense of ownership over the work done there. Both VSCode and Colab offer color themes for tuning the editing experience. Here, VSCode's extensions are an accessible plus, with easy opportunities to try many options. VSCode-pets are an especially wonderful option[7]! Colab requires extensions for colorful skinning and has built-in surprises like power mode, a "Colab crab," (with Corgi and Kitty siblings, wandering the menu bar), and the marquee function. These allow additional connections between professors/TAs and students; they add elements of fun and elements of control for the students when learning CS might seem unduly unnatural *On this axis:* **Toss Up**

4.2 Suitability for Intro Computing

We have encountered strong student sentiment that College Computing/ Comp1/Comp2 courses should build different skills – and should *feel* different - from learning-to-code sites like Scratch and Tynker sometimes encountered in (pre)secondary settings. We find students receptive to required computing, as long as the toolsets and skillsets employed are **authentically representative of today's downstream work environments** along post-comp1 paths. Indeed, VSCode represents the consensus look-and-feel: its interface is the go-to background for media conveying "software development." What is more, VSCode holds today's plurality (though not majority) IDE use[5]. Colab, while not reminiscent of any particular learn-to-code site, is closer to pre-college experiences: its "cushioning," interleaving context with code cells, is popular in many data-science and natural-science contexts. We believe that Colab will be the look-and-feel of the future. For the present, *on this axis:* **VSCode**

That "cushioning" has a deeper implication: Colab does not offer students hands-on access to their own filesystems (or a *local* filesystem, in any case). VSCode provides access - in fact, insists on access - to the local filesystem, a conceptual model that is valuable *for some future paths*, though not only the ones with “computational" in their name. In short, VSCode offers a path into the universe of computing resources; Colab offers a well-padded abstraction of them. This is where audience-focus will likely carry the day: at our consortium, it’s where the largest differences hinge. *On this axis:* **Toss Up**

In a large class, it is important to consider the difference between presenting and creating results. Instructors seek to stage demos smoothly, which is where Colab shines - it is easy to set up and present results. However, when creating entire projects, VSCode wins. By having explanatory instructions intermixed with the executable code, Colab can become convoluted and confusing in demonstration settings. VSCode offers the context in a tab or panel *beside* the execution, a juxtaposition our students have come to expect . It is difficult to declare a clear winner here, as Colab is better for *reading* results, and VSCode shines when *presenting* results. *On this axis:* **Toss Up**

4.3 Logistical Axes: Keeping things running...

Whatever future workplaces might use, the constraints of the academic environment are important to instructors here-and-now. **Grading**, regardless of the philosophy or norms by which it is pursued[2], is a necessary part of a Comp1/CS1 experience. How do each of these platforms facilitate grading? Having edited and executed within VSCode, our students typically upload .py files for review. Autograders are used to grade some assignments, but far from all. As Colab is online and can be updated - and entirely lost, as does happen - we have learned to ask students to downloaded a snapshot of their notebook and submit that, instead of only its URL. This has the added advantage of maintaining a local copy on their own machines! (and ours!) Further, our Comp1 intersperses reading-responses and other types of content-creating alongside the software: Colab makes this natural. *On this axis:* **Colab**

Our Comp1 does ask students to run executable artifacts from their own machines. VSCode **installation and upgradability** has varied in ease over many semesters, but it has never been painless. Maybe this is a shared experience of frustration, unifying students and instructor against a common enemy, or maybe it’s time poorly spent. Either way, Google Colab doesn’t require setup, and students can familiarize themselves with Colab as they work through their first few computational challenges. *On this axis:* **Colab**

ChatGPT and other **AI-assistants**, e.g., github copilot, are currently better integrated within VSCode. Given the uncertainty around their incorporation into academic courses, this may be a plus or a minus. What’s more, we suspect that sooner rather than later, AI-assistance will be normalized, available, and configurable in both IDEs. For the moment, *on this axis:* **VSCode**

4.4 Commercial Concerns

VSCode and Google Colab represent not only two different companies, but two distinct **commercial ecosystems** and philosophies re: computing's future role. Microsoft owns VSCode, whereas Google owns Google Colab; Google seeks to become the computational engine that builds bridges from all disciplines' day-to-day work (via Drive and Docs) to support instances where scripting can help (via Colab). VSCode is converging to the same place, but from a software-centric starting point, expanding to embrace their users as they find themselves, perhaps unexpectedly, in the role of computational exploration. Bottom-up may not be how educational institutions are organized, but it *is* how education, and especially GenEd, is experienced: *On this axis:* **Colab**

Where the commercial ecosystems are least hidden is in the editors' **default behaviors**, which are especially important in students' earliest experiences. Both Colab and VSCode target an audience with experience, for example, with default popups that explain the parameters of built-in functions, such as print. Pedagogically, these are authentic, but unhelpful. The silver lining to their insistent attention-grabbing is that the vast majority of students automatically and unconsciously tune them out.[2] VSCode profiles allow instructors to smoothly customize their students' environments. Colab would benefit from a similar capability. *On this axis:* **VSCode**

Accessibility and price are crucial considerations. Not only is the price important when considering requiring a large number of students to use a particular platform, but it is also important after the class ends: an expensive toolset is less likely to be woven into future pursuits, even if it might help[1]. Both VSCode and Google Colab are free, for the moment. Because of their value to each titan's ecosystem, it seems likely they will be free for the foreseeable future. Both offer paid versions and extensions, whose capabilities are not important for either a GenEd introduction or, in our experience, anywhere in the undergraduate curriculum. Rather, they are far more likely to serve as a natural and familiar bridge to additional *compute-resources*, if - or when - those might be needed. The *"...you're the product..."* quip does apply, more for cloud-based Colab than locally-run VSCode. Yet fifteen years of that quip's normalization leads us to conclude, *on this axis:* **Toss Up**

5 Verdict and Evolution

If we were keeping score across the prior section's twelve axes, for our institution and cohorts, VSCode emerges the better choice on three of them, Colab on four of them, with “Toss Up" the most common categorization!

[2]Indeed, watching these message-boxes consistently fail to make it beyond their viewers' retinas is one of the silver linings of their otherwise depressingly quotidian presence.

For our situation, the experiment with using both Colab and VSCode for our GenEd Comp1 and its successor Comp2 has resulted in the following outcomes within our consortium:

- The Comp1 taken by all undergraduate students *across all academic disciplines* is best served by using Colab as its "computational hometown." Colab's ease of access, immediacy of exploration, and structural support for interlacing computing with context, explanation, and reflection – all of these well serve the foundational engagement the institution and students seek, regardless of the details of their future paths. For these students, Comp2 is also best served with Colab - or another jupyter notebook scaffolding, in order to build on the foundation established in Comp1. (VSCode offers an exceptionally capable environment for jupyter-notebooks; Comp2 can be horizon-expanding in platform, as well as computationally.)
- For the same reason, the Comp1 and Comp2 taken by our consortium's masters-degree students and other graduate students – in nominally non-cs disciplines – is also best served by a Colab-based introduction. In those programs, computing is not part of the program's identity, but a potentially valuable resource, brought to bear on problems and tasks insofar as it adds value. Here, facilitating and contextualizing exploration are the crucial criteria.
- On the other hand, the Comp1 taken by all students across *all STEM academic disciplines* is better served by a hybrid set of experiences. VSCode is our initial foundation for local file-interaction and execution, supplemented later with Colab-based explorations where that platform better supports exploration (e.g., turtle graphics) or when communicating contextualized results. A mix of environments is a challenge, but for a STEM cohort it is a worthwhile challenge *per se*: we hear from our sibling STEM departments that, because they already leverage so many different systems, their students benefit disproportionately from the adaptability that comes with successfully picking up new computational-authoring environments and solving problems in them. This is precisely our approach - and philosophy for Comp1/Comp2. (Incidentally, it's also our approach for CS1/CS2 and all the rest of our CS curriculum.)

It is not a surprise that one size does not fit all: there is no need!

All deliberately scaffolded introductions-to-computing have the opportunity to be positive. The considerations here have helped us converge on this moment's deliberate approaches, as we seek to make computing's Cambrian explosion as comfortable and positive as possible, for our students and ourselves.

6 Perspective

It would seem there is a verdict, but that the verdict is less about the "right" choice of computational IDEs and more about how rapidly computing's role is evolving in higher education. This work's taxonomy has been prompted, in part, by the momentum our institutions sense, both bottom-up and top-down, behind making Comp1 a universal General Education requirement.

As more institutions and students make computational authorship part of the fundamental literacies students practice in their college/university experience, the IDEs they use will continue to mature. The axes outlined here, we hope, will help assess such resources as we all respond, proactively, to the changes before us.

As computing instructors, we have the opportunity - and responsibility - of choosing our students' computational birthplace. Let's choose wisely!

Acknowledgements

The authors gratefully thank and acknowledge the support of NSF CISE Project #2142780 and support from Harvey Mudd College.

References

[1] S Bhattacharyya. "Is Matlab losing its Charm?" In: Analytics India Magazine (2022). `https://analyticsindiamag.com/is-matlab-losing-its-charm/`.

[2] Dan Garcia et al. "Achieving" A's for All (as Time and Interest Allow)"". In: Proceedings of the Ninth ACM Conference on Learning@ Scale. 2022, pp. 255–258.

[3] Google Colab. `https://colab.research.google.com/`.

[4] Live Share. `https://code.visualstudio.com/learn/collaboration/live-share`.

[5] Top IDE Index. `https://pypl.github.io/IDE.html`.

[6] VSCode. `https://code.visualstudio.com/`.

[7] VSCode-Pets. `https://marketplace.visualstudio.com/items?itemName=tonybaloney.vscode-pets`.

[8] World Happiness Report. `https://worldhappiness.report/`.

An Introduction to Java OpenMP Parallel Programming with omp4j

Conference Tutorial

Xuguang Chen
Department of Computer Science
Saint Martin's University
Lacey, WA 98503-3200
xchen@stmartin.edu

Because multi-core systems have become more and more popular in various application areas, how to leverage multiple processors has become a problem that must be answered. Parallel and distributed programming skills traditionally are a topic in a High-Performance Computing (HPC) elective, and however, have started to be integrated into early CS curricula in recent years.

General speaking, parallel programming models can be classified into two categories: the message passing model and shared memory model. Open Multi-Processing (OpenMP) jointly defined by a group of computer hardware and software vendors is an application programming interface (API) supporting shared memory model. It can be implemented in various programming languages, such as C, C++, Java, Python, and FORTRAN, and can be executed on many operating systems, such as Windows, Linux, macOS, and Solaris. OpenMP has many advantages. For example, the responsibility of creating and managing thread traditionally completed by the programmer was removed. Instead, abstract and preprocessor level directives and clauses in its API are used by the programmer to write the code performing parallel computing. Correspondingly, OpenMP can make it easier for students to write parallel code, because students in class can focus more on the concept of parallel computing rather than low-level threaded code.

Java is an object-oriented and general-purpose programming language having been used in many areas, including parallel computing. The omp4j is one of the attempts bringing OpenMP to Java. It is a lightweight Java OMP-like preprocessor and written in Java and Scala. This tutorial will focus on the basic skills conducting Java OpenMP parallel programming using omp4j. At first, the tutorial shows where to acquire omp4j software and how to use it on Windows machines and Linux machines. After that, it covers how to edit, compile, and run an Open program on a Windows machine and a Linux machine. Then, the applications of several basic OpenMP directives are explained, followed by hand on examples. Finally, the sample code used in the tutorial will be provided and other materials suitable for self-study will be introduced.

The tutorial is focusing on the audience who is a beginner to parallel programming and/or is interested in OpenMP programming, having basic knowledge of the programming languages, such as Python, Java, C#, FORTRAN, and/or C++. The expected learning outcomes are the followings. After attending the tutorial, the audience should know where to get a copy of omp4j and how to use it on a windows machine or Linux machine. Moreover, the audience will also know the concepts of shared memory model, and the difference between the message passing model and shared memory model. Other than that, how to edit, compile, and run an OpenMP in Java will be learned. In addition, the audience should learn how to implement various basic OpenMP operations with omp4j. At the end of the tutorial, the audience can be provided the e-version of the lecture notes, code as examples, and other materials for self-study, if needed.

References

[1] omp4j. `https://github.com/omp4j/omp4j`.

[2] OpenMP Tutorials. `https://www.openmp.org/resources/tutorials-articles/`.

[3] B Barney. OpenMP Tutorial. `https://hpc-tutorials.llnl.gov/openmp/`.

[4] Petr Bělohlávek. OpenMP for Java. 2015.

[5] Robert P Cook. An OpenMP library for Java. In *2013 Proceedings of IEEE Southeastcon*, pages 1–6. IEEE, 2013.

Getting Started on Jetstream2*

Conference Tutorial

Zachary Graber, Daniel Havert
Research Technologies
Indiana University
Bloomington, IN

As research and education advance, so does their need for advanced computational resources. While some universities are fortunate to be able to provide these resources in abundance, many may not have free availability to such cyberinfrastructure for their research, much less for their instruction. Through Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS), advanced computing resources are being shared with educators for free, among them Jetstream2. This sharing of resources provides access to educators who normally would not have access to such platforms.

Jetstream2 is an NSF-funded user-friendly cloud computing environment for researchers and educators running on OpenStack and featuring Exosphere as the primary user interface. It is built on the successes of Jetstream1 and continues the main features of that system while extending to a broader range of hardware and services, including GPUs, large memory nodes, virtual clustering, and many other features. It is designed to provide both infrastructure for gateways and other "always on" services as well as to give researchers and educators access to interactive computing and data analysis resources on demand. One of the goals of providing such a resource without cost is to be able to provide colleges and universities access to these resources not only for research but for instruction, thereby democratizing cloud computing for educators.

Tutorial Proposal

This tutorial targets an audience of educators, researchers, and IT pros. Attendees will get an overview of Jetstream2, the ACCESS ecosystem and how to get on Jetstream2 then will be walked through how to access and launch

*Copyright is held by the author/owner.

VMs on Jetstream2 through the Exosphere interface, provide various examples and use cases of Jetstream2 for instruction, along with other helpful tips and tricks.

Requirements

- An ACCESS account. Can be created for free at https://identity.access-ci.org/new-user.
- Also, please let us know your ACCESS username so we can add you to a special training allocation so you can follow along with the tutorial: https://forms.gle/WZRNMrdkratTLZ8L8
- A computer with internet access.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant 2005506. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Biography

Zachary Graber is part of the Research Cloud Services team in the Research Technologies division of Indiana University (IU) that supports Jetstream2. He received his bachelor's degree in Computer Science from IU's Luddy School of Informatics, Computing, and Engineering.

Daniel Havert is part of the Research Cloud Services team in the Research Technologies division of Indiana University that supports Jetstream2. He received his bachelor's degree in Physics from Embry-Riddle Aeronautical University and is currently completing a PhD in Physics at Indiana University. His interests include cloud computing, artificial intelligence, and educational outreach.

References

[1] David Y Hancock, Jeremy Fischer, John Michael Lowe, Winona Snapp-Childs, Marlon Pierce, Suresh Marru, J Eric Coulter, Matthew Vaughn, Brian Beck, Nirav Merchant, et al. Jetstream2: Accelerating cloud computing via jetstream. In *Practice and Experience in Advanced Research Computing*, pages 1–8. 2021.

Building and Using a Penetration Testing Environment of Virtual Machines*

Mohamed Lotfy
Utah Valley University
Orem, UT 84058
MohamedL@uvu.edu

To prepare IT and cybersecurity graduates and meet industry needs, cybersecurity courses must introduce current offensive and defensive tools and practices. Teaching offensive security (penetration testing/ethical hacking) is becoming a standard practice in computer science, cybersecurity, and information technology programs[2, 5]. Penetration testing/ethical hacking allow students to perform a sequence of different phases to gain the needed cybersecurity knowledge and skills using current tools. These current offensive cybersecurity tools should be introduced and applied in the different hands-on activities thus allowing students to gain the needed knowledge of current cybersecurity best practices[2]. Through a hands-on approach, penetration testing/ethical hacking courses allow students to develop offensive cybersecurity competency enabling them later to build layered defenses that hardens the systems to penetration. Teaching penetration testing requires an attacking host that is used to perform the different phases of penetration testing on vulnerable hosts. Using an encapsulated virtual environment where the different attacks on vulnerable hosts can be conducted, reduces the risk to institutional networks and systems. Attendees will exit the tutorial with an idea of how to create a working VMware or VirtualBox environment and learn how to perform some of the phases of penetration testing using a Kali Linux attack host.

Tutorial Description

In this tutorial the presenter will provide an example of how to create a penetration testing environment using VMware Workstation Pro and Oracle VM

*Copyright is held by the author/owner.

VirtualBox on laptops or PCs[4, 3]. The virtual environment will include an Offensive Security Kali Linux VM, a Metasploitable Linux VM by Rapid7, a Metasploitable Windows 2008 server, and a customized Windows XP VM.

In the tutorial the following will be demonstrated:

1. The structure of ethical hacking virtual environment using multiple hosts.
2. Offensive tools on Offensive Security Kali Linux.
3. Port and operating system scanning using Nmap.
4. Exploitation using Metasploit Meterpreter.

Target audience

Any faculty who desires to incorporate a virtual environment and use it in a penetration testing/ethical hacking course. Attendees should be familiar with Linux, networking, and some programming knowledge. It is highly recommended that attendees bring their own laptops with VMware or VirtualBox and a Kali Linux VM installed [1].

References

[1] OffSec Services Limited. KALI pre-built virtual machines, 2023. URL: `https://www.kali.org/get-kali/\#kali-virtual-machines`.

[2] Mohamed Lotfy. Teaching a penetration testing course during covid-19: lessons learned. *Journal of Computing Sciences in Colleges*, 37(2):80–88, 2021.

[3] ORACLE. ORACLE VM VirtualBox, 2023. URL: `https://www.virtualbox.org/`.

[4] VMware. VMware Workstation Pro, 2023. URL: `https://www.vmware.com`.

[5] Xiaodong Yue and Hyungbae Park. Design of virtual labs for an ethical hacking course. *Journal of Computing Sciences in Colleges*, 35(6):31–38, 2020.

Hands-On Circuits in CS1[*]

Conference Tutorial

Elena Miller, Katy Shaw, Zachary Dodds
Computer Science Department
Harvey Mudd College
Claremont, CA 91711
{elmiller, kshaw, dodds}@hmc.edu

1 What is computing made of, these days?

Many introductory computer science courses - including ours - focus on the science and art of programming. In such classes - including ours - the predominant activity is editing: editing one's electronic documents and editing one's conceptual models of computation. Both of these important skills rest on the foundation of a conceptual model of computation's physical processes.

Today's computing interactions rightfully abstract away the switching of transistors and the cascade of logical circuits. As CS1 finds itself a GenEd requirement for more and more students and institutions, as is true for us, experiencing its bigger-picture connections becomes more important. To make the physical underpinnings of computing accessible to all students, many of whom will never again touch a transistor or logic gate, we have developed, tested, refined, and deployed a short activity accessible to all students, regardless of background. This tutorial invites its participants to try out these activities and materials.

Participants in this tutorial will work through our "transistor and logic gates" lab, in which they will create (1) a NOT gate from two transistors and (2) an arithmetic circuit from logic gates. Both are accomplished with low-cost, off-the-shelf parts, totaling about $10. No specialty equipment - no scopes or generators - are involved, and the materials are extremely portable. Through the tutorial, participants will (re)experience setting up their own breadboard and assembling resistors, transistors, wires, and LEDs.

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

As a GenEd, our course includes assignments not present in every Comp1/CS1, e.g., hands-on assembly language, (simulated) circuit design, and several "non-programming" computational models, e.g., FSMs and TMs. We will share how we integrate this hands-on lab so that prior experience is neither necessary nor expected – and such that no future experience is anticipated. We will share, too, the online resources we find complement these paths, e.g., CircuitVerse and the NAND Game [1, 2].

We are grateful to stand on the shoulders of prior work demonstrating the value of hands-on circuit design [4]. In addition, we acknowledge the ambition and scope of courses, such as From NAND to Tetris, that center the technology stack that is modern computation [3]. For us, practicing software-authoring is of paramount importance for all students, so this tutorial's more modest approach works well.

In such a spirit of adaptation, we will be delighted if participants adapt this tutorial's materials to suit their home institutions' computing pathways. Plus, whether adaptation is possible or not, we are excited for all participants to have built a logic gate from physical transistors and an arithmetic circuit from physical logic gates – and to encounter how accessible such experiences can be.

Acknowledgements

The authors happily and gratefully thank HMC and the NSF: CISE #2142780.

References

[1] CircuitVerse - Online Digital Logic Circuit Simulator. `https://circuitverse.org/`.

[2] The NAND Game. `https://nandgame.com/`.

[3] Shimon Schocken. Nand to Tetris: building a modern computer system from first principles. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 1052–1052, 2018.

[4] Hsien-Tsai Wu, Po-Chun Hsu, Chih-Yuan Lee, Hou-Jun Wang, and Cheuk-Kwan Sun. The impact of supplementary hands-on practice on learning in introductory computer science course for freshmen. *Computers & Education*, 70:1–8, 2014.

Reviewers — 2023 CCSC Northwestern Conference

Xuguang Chen	Saint Martin's University, Lacey, WA
Haiyan Cheng	Willamette University, Salem, OR
Janet Davis	Whitman College, Walla Walla, WA
Zachary Dodds	Harvey Mudd College, Claremont, CA
Radana Dvorak	Saint Martin's University, Lacey, WA
Markus Geissler	Cosumnes River College, Sacramento CA
Robert R. Lewis	Washington State University Tri-Cities, Richland, WA
Razvan A. Mezei	Saint Martin's University, Lacey, WA
Ben Tribelhorn	University of Portland, Portland, OR
Tammy VanDeGrift	University of Portland, Portland, OR
Howard Whitston	University of South Alabama, Mobile, AL