Keeping Mindful of Modality: A Comparison of Computer Science Education Resources for Learning

Michael J. Johnson
michael.johnson@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Rachel Baker-Ramos
rachelbaker@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Christopher Lynnly Hovey
hoveyc@colorado.edu
University of Colorado Boulder
Boulder, Colorado, USA

Betsy DiSalvo
bdisalvo@cc.gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

ABSTRACT

Educators often use computer science education resources to enhance the learning process, which come with a variety of output modalities (e.g., audiovisual, tangible) and coding modalities (e.g., block-based, text-based). While these resources are typically evaluated for their applicability and impact on young populations, they are not often analyzed through a comparison of their coding and output modalities, nor as a whole to each other. In this paper, we conducted BridgeUP STEM, an afterschool CS course for high school women and gender non-conforming individuals aimed at developing their computational thinking skills and exposing them to coding and CS. We collected and analyzed interview data for 16 participants on their experiences within the course and attitudes towards various output and coding modalities. Throughout the study, the students' reflections on their own learning revealed the affordances and drawbacks of each resource in terms of outputs the resources provided, feedback the students received, and how both affected the students' troubleshooting. We present these findings and use them to provide recommendations for approaches to teaching computer science.

CCS CONCEPTS

• **Social and professional topics** → **Informal education**; *K-12 education.*

KEYWORDS

physical computing, screen-based computing, modality, computer science education, afterschool

ACM Reference Format:
Michael J. Johnson, Rachel Baker-Ramos, Christopher Lynnly Hovey, and Betsy DiSalvo. 2023. Keeping Mindful of Modality: A Comparison of Computer Science Education Resources for Learning. In *23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23), November 13–18, 2023, Koli, Finland*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3631802.3631819

Koli Calling '23, November 13–18, 2023, Koli, Finland

ACM ISBN 979-8-4007-1653-9/23/11.
https://doi.org/10.1145/3631802.3631819

1 INTRODUCTION

Designing innovative methods for introducing computer science (CS) to young populations continues to be a prominent area of CS education research. In recent years, there has been renewed attention and excitement around teaching CS using tangible artifacts that combine programming, electronics, art, and other functionalities. This computing sub-area, broadly known as *physical computing*, provides a different dimension to solving CS problems and an alternative to text-based coding tasks traditionally used in basic and advanced computer science education (CS ed). The use of physical computing is part of broader efforts in the education community to better introduce novice learners to computational thinking: the attitude that involves solving problems, designing solutions, and troubleshooting in ways fundamental to understanding computer science concepts [28, 37]. Research on incorporating physical computing into the classroom spans decades of growth and development of "low-floor", "high-ceiling", and "wide-walls" initiatives to get young learners interested in computing [27]. Physical computing is reported to increase motivation as the learning experience and outcomes are "visible, not virtual"; a visibility that further facilitates connections and engagement for learners [12]. To evolve past "traditional" CS ed practices, many interventions utilize a range of modalities that determine the medium and manner of input and output interactions with the user. This range of modalities demonstrates that there is likely no clear, single "best" alternative to traditional text-based programming. For example, several non-tangible, screen-only resources support the cultivation of students' CS programming skills and computational thinking.

Despite this diversity of modalities, CS ed research has focused on assessing if an intervention's content establishes computational thinking skills, with little consideration of the modalities used. Most past literature that has investigated the benefits of incorporating physical computing into educational contexts does so without comparing it to existing computing education methods [32]. In response to this gap in the literature, this paper's research focuses on differences between modalities to unpack nuances they provide for learning CS, emphasizing understanding how modality impacts

problem-solving. We propose that exploring CS ed can identify the affordances and drawbacks of modalities, and that understanding these factors is key to implementing better CS ed experiences for learners. In this paper, we explore a study conducted on the first cohort of BridgeUP STEM (BUS)—an afterschool CS course for high school women and gender non-conforming individuals—to compare how different modalities of CS activities were experienced by high school students. Our analysis is guided by the following research questions:

RQ1. What drawbacks and affordances can be identified from examining CS activity output modalities?
  (a) How does modality affect a learner's conceptual models of CS principles?
  (b) How does modality affect a learner's troubleshooting process?

RQ2. How can an understanding of modality contribute to improved CS experiences for learners?

Throughout the program, students interpreted feedback, debugged errors, and reflected on their skills as programmers to varying degrees. Variations in success appeared to stem from differences in coding and output modality. We aim to uncover the meaning behind these differences to understand and leverage them toward improving CS learning. Finally, we recommend several beginning courses of action the CS ed community can take to further develop computing education activities for broader use and reach.

1.1 Definitions

It is necessary here to clarify what is meant by the terms 'coding modality' and 'output modality'. We define *coding modality* as the means by which writing code is done. In BUS, this was either text-coding or block-based coding, sometimes referred to as the format of the coding editor (e.g., [15]). *Output modality* is the medium through which output is given, which is subject to the materials used. Output modalities not only return information to a user or enact a set of behaviors outlined in the program but also convey information about a code's success. An example of the latter is progress notifications, such as error messages printed to the console, a browser pop-up window, or a physical light blinking to indicate code was uploaded.

Students' reflections on these modalities often referred to 'output' and 'feedback' synonymously, and prior literature does not offer a distinct definition of either [8, 31]. However, for clarity, this paper will distinguish the difference between the two terms to help avoid them being conflated. We define feedback as one aspect of output modality, which individuals determined. In other words, *feedback is information provided by the output.* Additionally, a single output and its feedback are not mutually exclusive. For example, think of a cellphone receiving a phone call. The phone might ring and display an "incoming call" notification on the screen. Both outputs, the audio ring and the visual display, convey the same feedback to a user: there is an incoming phone call. Likewise, two individuals may both hear the same phone ring, with one supposing it is an incoming call and the other supposing a timer may be going off. In this example, the same output leads to different feedback per person. As we proceed in this paper, we will continue to distinguish between output and feedback while discussing our findings.

2 RELATED WORK

2.1 The Current Landscape of Physical Computing Education

While the focus of this paper is to explain the comparison of feedback modalities beyond those found in physical computing, some background in the rise of the physical computing education subfield contextualizes its influence on computer science education and where future research is needed.

Physical computing has received multiple definitions since its inception. It has sometimes referred to the process of designing interactive, electronic objects that can communicate with humans via sensors and actuators controlled by programmable hardware [3, 24, 25, 30]. That process is often discussed in the context of CS education, where those interactive objects and their corresponding software are used to teach students computational thinking skills with hands-on activities [17, 18]. At other times, physical computing is used as a term to encompass the programmable interactive objects themselves [12, 24]. Early work by Marshall [21] on the benefits of physical computing provided an analytic framework of six perspectives that might guide research on tangible interfaces for learning. Of these were possible learning benefits afforded by these interfaces, which included: the use of physical materials might change the nature of knowledge gained compared to virtual materials; tangible interfaces may be more suited for engaging children in playful learning with a physical action that causes digital effects; and collaborative learning is well-supported by group work with tangible interfaces. Still, the relative benefits of combinations of concrete and abstract representations regarding physical components have yet to be investigated [21]. In addition, Hodges et al. [12] summarizes the benefits of physical computing as seen through their experimentation with the BBC micro:bit [2] in the classroom. They generalized their findings to all physical computing devices to claim that physical computing increases motivation because the learning experience and outcomes are "visible, not virtual," and that tangibility yields natural connections for learners. They also stated that the interactivity of physical computing leads to a better understanding of programming concepts when debugging, as the multiple potential solutions to a problem promote a trial-and-error approach. Other reported benefits include students exhibiting higher creativity in what they build, and strengthening engagement with the physical task. These types of activities then open the door for more collaborative practices in group work [12].

Some studies have focused on how incorporating physical computing into curricula may affect educators. Sentance et al. [30] investigated the use of micro:bits for classroom instruction and culminated their findings into providing recommendations to support teachers better. They emphasized that teachers need prep time to add physical computing to a curriculum, and should be provided with professional training in using those devices themselves. Sentance et al. also stated that researchers or designers of educational interventions should try to consult with 'expert' teachers who can develop teaching material with strong pedagogical approaches. They suggest that educators have projects spanning multiple weeks with supporting assessments linked to the curriculum, which will be better than short, un-sequenced activities taking place over smaller amounts of time [30]. Kalelioglu and Sentance

[14] explored teachers' pedagogies and strategies for using physical computing in their curriculum. Teachers shared responses indicating themes of easy use for students, with higher motivation being attributed to receiving "immediate feedback". Themes of the potential for enhanced feedback from tangible devices have also made way for robotics to be researched as learning objects with high potential for changing teaching and learning practices [7]. Many of the above studies agree upon these benefits. However, these studies, and those from other literature, generalize too much on "physical computing" as a whole. Instead, we aim to take a nuanced look at its components for further insights.

2.2 Perceptions of Feedback and Errors

The importance of feedback in computer programming has mostly been explored in terms of how users interact with programming error messages (PEMs), and with good reason. PEMs are often the most informative output reflective of the status and failure of a program, which in turn supply users with feedback towards debugging issues [33]. Users not only read PEMs, but also spend a significant amount of their time doing so when debugging code [4]. Therefore, they become a major, though informal, teaching tool for those learning CS programming, or as Becker et al. [5] put it, "Programming error messages are pedagogically important, particularly in their roles as feedback agents. This is unlikely to change for some time." Prior research on the challenges faced by novice programmers interpreting compiler error messages dates back to as early as 1965 [5] and continues to this day. Recent efforts exploring the helpfulness of PEMs have resulted in numerous guidelines for their improvement, summarized by Becker et al. [5]. Of these guidelines, increasing the readability of the errors has been mentioned as a prominent step towards clearly conveying their meaning. Denny et al. [8] followed up on this topic by developing four concrete design guidelines to increase readability in error messages: 1) use commonly understood words (i.e., remove jargon), 2) write messages in complete sentences, rather than strings of broken words and symbols, 3) use simple vocabulary, and 4) use no more or fewer words than necessary to get the point across.

Notably, much of the work in improving the interpretation of error messages has been done in instances where compilers print out errors to a console. As this paper will discuss, there are many potential output modalities that convey a program's success or failure. There has been some work in robotics that has looked at tangible feedback and how it could be more impactful than virtual work [25], but perceptions of feedback and errors remain underexplored when accounting for methods beyond console output.

2.3 Modalities of Computer Programming

Research into coding modalities has often highlighted the differences between block-based and text-based programming. Studies show that block-based editors help K-12 students develop computational thinking concepts and practices, but fail to scale to larger programs [11, 15, 26, 36]. Users also report that block-based editors feel less authentic and take more time [10, 15, 35]. Comparison work has found differences in learning gains [36], ease-of-use [35], and retention of information post-transition from blocks to text [15]. On a more abstract level, there has been research investigating the transparency of programming modalities [22] and in physical computing toolkits [9]. However, these studies have not looked at the output given in a computing context. Given the prevalence of literature comparing other pedagogical methods of CS ed, we are suggesting more research be conducted in comparing output modalities.

One notable paper that begins to address this gap in the literature is a recent study by Love and Asempapa [18] that took an attitudinal approach to evaluate the differences between screen-based and physical computing activities. The study compared two groups learning CS. The first group employed Scratch for a game design unit. The second group used the Crumble microcontroller and programming platform to create vehicles for a collision avoidance design challenge unit. Both units were designed to cover basic programming skills (e.g., variables, loops, and functions), but differed in what the students themselves designed. The authors found that students in the screen-based unit group exhibited stronger attitudes (indicating a favorable disposition) towards coding in the classroom and applicability in their career/future than those using physical computing, even when controlling for prior engineering or CS coursework. Including those with this perceived advantage, the majority of their students felt learning with physical computing would make learning CS easier. While some students expressed concern about using hands-on devices that could malfunction or be difficult to troubleshoot, a similar percentage of students expressed that the hands-on experience was beneficial for providing "real-time tangible feedback". Surprisingly, the two groups had no significant difference in interest or comfort in coding. Instead, their findings suggest more nuance about the modality used.

This research corroborates our own efforts in recognizing the nuances of different output and coding modalities. It also suggests that more exploration is needed on leveraging affordances for physical computing while considering the advantages of screen-based interactions, which are far more common in real-world computing. Our study expands on Love and Asempapa's work by seeking to identify the benefits each modality offers and explore how to balance affordances and address the shortcomings of each modality.

3 CONTEXT

3.1 BridgeUP STEM Overview

BridgeUP STEM is a program conducted in partnership with the National Center for Women & Information Technology (NCWIT) and sponsored by the Helen Gurley Brown Foundation. In it, women and gender non-conforming high school students are exposed to collegiate-level research practices while learning CS at a research university. BUS was piloted at the Georgia Institute of Technology during the 2021-2023 school years. We recruited through connections with Atlanta-area high schools. We incentivized students to participate in BUS via $2,000 stipends distributed over the two-year program, contingent on having at least 90% attendance across sessions. Applications were open to all eligible students, promoted through their high school teachers, and made publicly available online on NCWIT 's website. Our goal was to recruit 20–25 students, with the expectation that some may drop off during the year. In 2021, 22 students applied to the program, all of whom were accepted and enrolled.

Table 1: Details about the five major resources used in the BridgeUP STEM programming course.

Unit	Resource	Materials	Coding Modality	Output Modality
Music-Mixing	EarSketch	mouse, keyboard, web browser	text-coding	audio, custom music, visual DAW representation
Artificial Intelligence	Jupyter Notebooks	mouse, keyboard, web browser	text-coding	console output, game board GUI (final project only)
Inputs & Outputs	Arduino Makeblock Kits	mouse, keyboard, Arduino IDE software, physical computing kits (sensors, motors, lights, etc.)	text-coding	physical interaction with sensors, lights, motor movement, Arduino serial monitor output
Complex Adaptive Systems	StarLogo Nova	mouse, keyboard, web browser	block-coding	GUI canvas, buttons, simulated movement, drawings
	MakeCode	mouse, keyboard, web browser, micro:bit, Yahboom line-following robot	block-coding	lights, robot vehicle movement

Participants for BUS were required to meet the following criteria, as outlined by the sponsor:

- Identify as a woman or non-binary individual
- Is 13 years of age or older at the time of application
- Is enrolled in 9th, 10th, or 11th grade as of the start of the program
- Is able to commute to the Georgia Tech campus approximately once or twice per week after school
- Have minimal or limited prior coding experience
- Is not an NCWIT or Georgia Tech employee or have an immediate family relationship with employees or either organization

Participants were also subject to additional tax reporting and legal requirements because they received a stipend.

In the program's first year, students attended a CS programming course taught at Georgia Tech. The course's goal was to introduce the students to basic computer programming skills through means determined by Georgia Tech facilitators. It was held as an after-school program that met twice weekly in 2.5-hour sessions from November 2021 to May 2022 (20 weeks).

3.2 Programming Course Overview

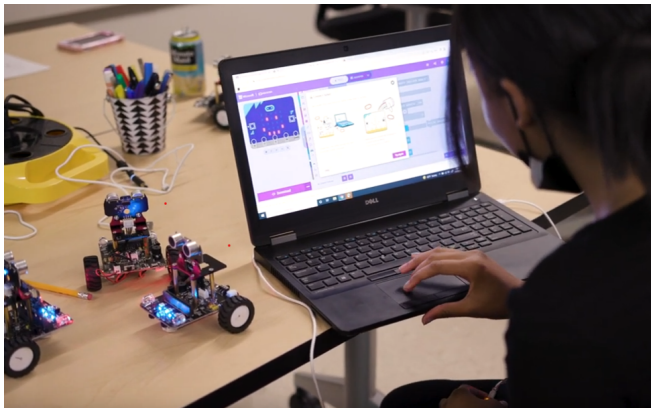
The Georgia Tech computing outreach department chose the first author of this paper to design and instruct the programming course. The first author served for six years as an in-person and online instructor for extracurricular computing courses at Georgia Tech, and was familiar with the staff and materials available for the course. These advantages allowed the course to be quickly constructed in a short time period. The first author is also a computer education researcher at Georgia Tech and designed the course to highlight the differences between output modalities in computing activities to then study with the students as research participants. Activities selected sought to expose the students to various computational thinking practices [34]. These included collecting and visualizing data, programming and debugging, developing modular computational solutions, and constructing models to understand concepts.

As mentioned in Section 3.1, students were expected to have minimal or limited prior experience with computer programming when enrolling in BUS, which resulted in a range of experiences among students. Therefore, all activities were planned to teach programming basics from the ground up, with room to challenge advanced students. The course contained four units, each meant to build on the next while exposing the students to a range of computational tools and resources in various modalities. The five major resources used in the units covered two major coding modalities: block-based and text-based programming. Table 1 shows an overview of each resource, the materials it used, and the associated coding and output modalities. The course syllabus can be found in Appendix A.1.

3.2.1 Music-Mixing Unit. The first unit introduced the Python programming language using EarSketch [19]. EarSketch combines computer programming with music mixing and creation in an online text-based coding editor. Written code populates an on-screen digital audio workstation (DAW) with music clips from an extensive sound library. Sound effects (e.g., volume gains, tempo changes, and other modulations) are applied by referencing the EarSketch API. EarSketch introduced Python basics—such as creating and running scripts, using variables and loops, and simple debugging—from a combination of the built-in EarSketch curriculum and the instructor's own experience. The CT practices covered by this unit included programming, developing modular computational solutions, and debugging.

3.2.2 Artificial Intelligence Unit. The second unit introduced artificial intelligence (AI). The curriculum was adapted from an IBM SkillsBuild course on AI [13], as well as the instructor's prior experience. Students were introduced to various types of AI and machine learning methods through interactive online resources, coding activities, and small-team research projects. Students used Jupyter Notebooks [16] embedded with instructions as the medium for all programming work. The CT practices covered by this unit included preparing problems for CT solutions, programming, assessing different approaches to a problem, and debugging.

(a) An interactive, Arduino Makeblock lamp created by P3 and P12 using LED lights, a sound sensor, and a motion sensor.



(b) A student programming multiple Yahboom line-following robots with MakeCode.

Figure 1: Images of projects from (a) the Inputs & Outputs unit and (b) the Complex Adaptive Systems unit.

3.2.3 Inputs & Outputs Unit. In the third unit, students used Makeblock Arduino, a physical computing kit containing numerous interactive electronic modules [20]. The modules attach to a modified Arduino Uno (called a Makeblock Orion) via RJ25 wires and are programmed in the Arduino IDE coding environment. Among the modules are several types of sensors, a joystick, LED lights, a servo motor, and a 7-segment display. As the Arduino programming language is fairly different from Python, the students relied on a set of example code provided by the kits and the instructor to understand their functionality. The instructor introduced each module individually, then revealed how input and output modules could be combined to interact with each other. Students then worked in small groups to design interactive objects with the kits and craft materials (see Figure 1a). The CT practices covered by this unit included collecting data, programming, choosing effective tools, and debugging.

3.2.4 Complex Adaptive Systems Unit. The final unit introduced the students to complex adaptive systems (CAS). CAS can be defined as multi-agent, leaderless environments where each agent individually acts out a similar set of commands without a centralized communication hub. The students experimented with these systems through two complementary activities. First, they used StarLogo Nova [23], an online programming editor, to create complex systems on a 3D canvas. The instructor followed the Project GUTS curriculum [29] to introduce students to programming in StarLogo Nova and enacting their own CAS for experimentation. The activities progressed from each other, ending in the creation of a pandemic outbreak model with integrated tools to monitor and change relevant variables. Students then programmed Yahboom line-following robot vehicles [38] to act as a physical representation of agents in a toy environment (see Figure 1b). These vehicles were controlled by BBC micro:bits [1] and programmed online with the MakeCode editor [2]. StarLogo Nova and MakeCode both support block-based programming, which exposed students to this coding modality for the first time in the course. The CT practices covered by this unit included collecting data, visualizing data, using models to understand a concept, constructing models, programming, and investigating a complex system as a whole.

4 METHODS

Of the 22 students enrolled in BUS, 16 opted in to be participants in our study. Enrollment to BridgeUP STEM was required to participate in our study, but participating in the study was not a requirement of BridgeUP STEM. Recruitment for participation in the study was handled on the first day of the course via word-of-mouth. Participants did not receive additional compensation for being a part of the study.

4.1 Sampling Frame

At the start of the program, 5 students (31%) were first years, 7 (44%) were sophomores, and 4 (25%) were juniors in high school. Slightly over half (9 students, 56%) had previously participated in at least one computer, technology, robotics, or programming club inside or outside of school, and all but two students (88%) have taken one or more computer science courses in middle or high school. The grade level makeup is visualized in Figure 2.

In line with the broader goals of this program, a majority of participants came from historically underrepresented groups in computing. We report demographics; however, differences between demographic groups were not a focus of our study. While gender or race in relationship to modalities of output and programming might have impacted students learning, we did not see evidence in our analysis. This may be an area for future study. All students who participated in the study identified as women/girls or non-binary/gender non-conforming. Ten (63%) participants identify as Black or African American; 3 (19%) as Hispanic, Latina, or Latinx; and 3 (19%) as white. All students were born in the United States, and 2 students (13%) reported that they speak English as a second language. When asked, "Are you living with a disability? This can be either visible or invisible," 5 students (31%) selected "I'm not sure" or left this item blank. Using U.S. census data and mapping home zip codes with median income data, 3 (19%) hailed from zip codes defined as low-income communities, 12 (75%) from

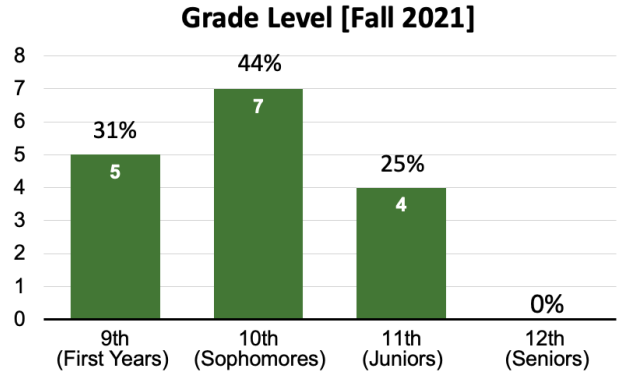


Figure 2: Grade level for BridgeUP STEM participants enrolled in the study.

middle-income communities, and 1 (6%) from a high-income community.[1] 12 students (75%) reported having at least one parent who had earned a four-year college degree, and 7 students (44%) reported that one or more parents had completed a post-graduate degree. The racial/ethnic identity and community poverty status makeups are visualized in Figure 3.

4.2 Data Collection

To fully capture students' thoughts and reflections, we conducted interviews before and after the course. Pre-interviews were conducted by multiple researchers following the same interview protocol and held during the first two weeks of the course while the students worked on the EarSketch unit. We gathered basic information about the students, such as their interest in the BUS program itself, their thoughts on technology, and their access to computing at home and school. The pre-interviews also captured students' attitudes and perspectives on computing and their plans for the future. The post-interviews were conducted by the first author and began the week after the course concluded. These interviews focused on the students' experiences from BUS. We asked about their favorite and least favorite activities, experiences working with other students, and how they would compare learning with certain resources to others. The post-interview also contained the same questions on the students' attitudes towards technology and computing and on their plans for the future. In addition to individual interviews, the first author conducted two rounds of focus groups throughout the course consisting of four students chosen by random selection. Three focus groups were held in the first round, while only two were held in the second round due to absences. Each focus group lasted approximately 30 minutes and asked the students to discuss their current thoughts on the course and to reflect on the activities, materials, and overall structure of the class.

4.3 Analysis

An automated transcription service transcribed audio recordings of the interviews and focus groups; then, researchers edited them for correctness. The first and second authors used thematic analysis [6] to analyze the transcripts. Thematic analysis consisted of an initial round of individual open-coding on a subset of the data. This was followed by the researchers drafting a codebook with an emphasis on how students interacted with each activity's materials, their sense of ownership of their work, and their troubleshooting processes. After a second round of coding, the researchers reconvened to revise the codebook, establishing categories of confidence and metacognition, as well as consolidating previous codes into an overall list of steps of problem-solving. After achieving inter-rater reliability of 85%, the finalized codebook was applied to all transcripts and coded excerpts extracted. Among the excerpts, we searched for notable common themes and selected to focus on output, feedback and troubleshooting. These themes are reported in Section 5 and further discussed in Section 6.

[1] A low-income community is defined as an area where the median household income is 80% or less of the metropolitan statistical area's median household income. A high-income community is an area where the median household income is 120% or more of the metropolitan statistical area's median household income.

5 FINDINGS

5.1 Student Reflections on Output & Feedback

Throughout the program, students consistently self-reported their experiences with the material interfaces. In line with the traditional "trial-and-error" pipeline, the output of CS ed tools provides the students feedback on success and failure. The knowledge gained from this feedback improves students' understanding of the system they are working with, of similar systems they have experienced, and of computer programming. While all students seemed to indicate that feedback was at the forefront of their minds, they held a wide array of preferences about the way feedback was observed, if at all, based on how the output of the system was delivered.

5.1.1 Screen-based Modalities. The standard output produced from screen-based resources is text, as most coding editors simply print words to the console. However, more expressive outputs, such as sound or visuals, are also common. In environments such as Jupyter Notebooks, the text-based coding cells produce more text when run, whereas programs like EarSketch produce music. Both forms of output convey vital information on the success or failure of the code, but they are fundamentally different.

In general, students responded more positively to resources that supported expressive and/or audiovisual feedback modalities, regarding them as more exciting and engaging. Two students, P1 and P2, each discussed how the visual and musical output of EarSketch stood out to them. P1 explained that she liked how the EarSketch interface displayed information similar to how physical modalities produced output: "*With EarSketch, I liked how it's kind of the same... there's the interface where you could see the music overlays and stuff, just kind of similar to how you could see what your code does to a robot with physical computing.*"

For P2, the musical output and the process of making music were more engaging than the text output of other screen-based activities. When asked how EarSketch related to her thoughts on text-based coding, P2 expressed a preference for the former: "*I think it was the output, like, 'cause we were able to make music with it. Whereas, you know, with the Jupyter Notebook, it was plainly text-based.*"

Similarly, in a focus group, several students shared that they found the visual output of StarLogo Nova to be engaging and understandable. P12 gave a specific example of how the visual output

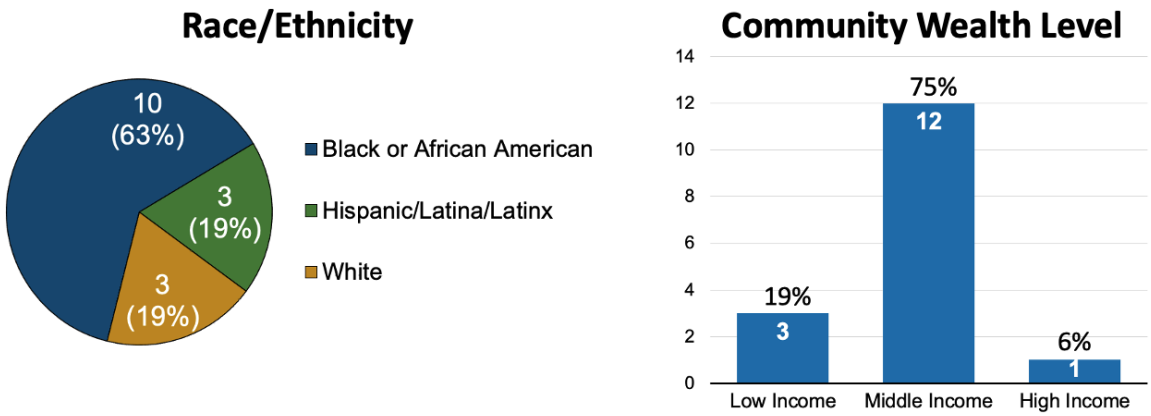


Figure 3: Race/ethnicity and community wealth of BridgeUP STEM participants enrolled in the study.

of StarLogo Nova assisted her understanding of the random function, stating that "*StarLogo Nova kind of helps [me] see it more, especially when we did like the bumper car ones when they changed color. And, like, I did one where I did like a random number of turtles on the screen at a time and it kind of like put into perspective like how the computer generates random [choices].*"

In contrast, students expressed dissatisfaction with screen-based resources that produced text output. P3 explained how text output can feel underwhelming and lead to a lack of fulfillment: "*...the ones that were just the coding part, you would finish the code and then you'd click 'run' or 'save' or whatever. And you just watch the little thing on the screen or hope that it [doesn't] give you an error.*"

P4, meanwhile, compared general scenarios of screen-based computing to physical computing, citing that in the former, "*what you do might not always be immediately seen.*" She found that gathering feedback from screen-based output was more time-consuming than gathering feedback from tangible output. This delay in feedback increased her frustration, making screen-based resources more difficult to work with. P4 later remarked that she would have liked screen-based computing activities to give more expressive output. She reflected on this statement regarding the AI unit and how it could have been improved: "*It doesn't have to be physical arts and crafts. In my mind, even doing something like, if we went back to the AI unit, creating an AI that could make art. I don't know if that's possible, but doing an arts and crafts like that—I feel like that would have made me even more excited.*"

While many students expressed that text-based output was not visually stimulating, one student ignored it altogether. P2 disregarded text as an output as it did not engage her senses. When asked for clarification, she responded that she doesn't view reading text as a sensory experience.

> **Interviewer:** "What does it mean to be output to you?"
> **P2:** "Being able to, I guess, use one of my five senses. Five senses, and for it to not be text."
> **Interviewer:** "Okay. So reading the text doesn't count as using that sense."
> **P2:** "Yeah."

Based on the definitions we are using throughout this paper, we find it is more appropriate to assume that P2 recognized that text was an output produced by some resources, but that she was unable to gain meaningful feedback from it to the point where it was almost as if the output did not exist.

5.1.2 Physical Modalities. With physical computing resources, feedback is front and center; the materials can produce a variety of output, such as light and movement, that convey a range of information. Many students found this output satisfying and engaging. P5 homed in on the tangibility afforded by physical computing materials, saying, "*I liked how we're able to actually, in-person, manually see our work come to life. And I think that's much more encouraging than just doing it on a computer. And maybe it says it works, but is not the same [satisfaction] when it is something you can touch or actually see happen in real life.*"

Parallel to the frustrations voiced with console text output (in instances where one cannot "see" the code), students explained that they could "see" the output with physical modalities. P1 explained that seeing the "*cause and effect*" with physical computing gave her more information than when only receiving it from the computer. Likewise, P6 spoke about the Yahboom robots and how she enjoyed observing the movements of the robots, stating that "*actually seeing*" the robot operate was "*cooler than just coding it.*". Other students such as P14, P7, P6, P10 and P13 echoed this sentiment during focus groups.

During her interview, P7's comparison of physical computing output to screen-based output devalued the latter, equating its feedback to nonsense: "*So it gives you something other than just, 'Oh, I can make the computer system print one plus one equals two.' It gives you something fun to make out of it and makes it just a little bit more enjoyable from a kid standpoint—doesn't really need it to complete a project or have a goal in the end; it's just to learn. So it made it more interesting because you got a more physical output than just a computer giving out a random print statement or number letter string, all that.*"

5.2 Student Reflections on Troubleshooting

Another prominent aspect of working with CS ed materials is how well they support the user when mistakes and errors occur.

Troubleshooting methods, often referred to as debugging, are an integral part of a student's problem-solving process. Feedback, as mentioned in the previous section, indicates to a user if their design was successful; troubleshooting methods indicate how a user addresses any issues. Just like feedback, these methods are subject to the resource's output modalities. However, we also found the coding modality to have a significant impact on troubleshooting and share those findings below.

5.2.1 Troubleshooting by Output Modality. Many students found troubleshooting easier with physical computing because of its tangibility and perceived rationality. P1 noted how the simplicity of observing tangible output, such as movements or lights, made it obvious which section of code was not working. *"If you see it, in-person, the wheels not moving correctly or the lights aren't flashing the way you want it to, that's a tangible thing you can look back at your code. That's why that's not working. Um, but with the computer coding, it's kind of different. It's not really feedback."* Most noteworthy here is the dismissal of feedback from a screen-based modality, just as P2 did when not recognizing any feedback from text output.

P8 shared a similar experience, explaining how she identified errors in her code faster with physical computing. She said the ability to "*actually see*" movement allowed her to easily pinpoint which section of code she should edit to fix a problem: "*Most times, you'll see a syntax error, but then you have to go and find the direct problem. But most times, when you actually see something move, then you'll know, 'Oh, maybe I have to go back to this section of the code.' So, I guess it's easier to find where the errors are.*"

P9 illustrated how physical computing often involved a rapid prototyping process, which allowed her to immediately spot errors. She compared this to her experience with screen-based computing, in which she felt that errors often occurred without indication, going unnoticed until later: "*The activities that were hands on, you were able to check them out on the computer... in real time actually see what was wrong and then change it. But online is more—you have one time to correct it. And if you mess up along the way, you wouldn't probably notice. Unlike when you're hands-on, you notice when you get it wrong the first time. And when and where.*"

P1 also described how physical computing allowed her to tackle troubleshooting incrementally, testing code step by step, making it easy to identify which section of the code needed to be edited. She compared this to Jupyter Notebooks, which she felt required her to complete her program before running it, making it more difficult to troubleshoot. She described how in the Jupyter Notebooks, "*you had to complete the code before you ran it more. Um, and you knew it wasn't working the way it was supposed to, but you didn't really know which step you kind of messed up on. With the physical computing, I felt it was more like you could make the code little by little and check at every step if it worked. And then run the code as a whole and you could troubleshoot from there. Because there's a lot of components to it that were separate with the physical computing.*"

In line with troubleshooting by visualizing issues, while working with the Yahboom robots, it was suggested to the class to display an icon on the LED matrix of the micro:bits each time they uploaded new code. This helped evaluate if the new code uploaded correctly without affecting the performance of the vehicle. P8 commented on this tip when discussing what she liked about the robots, saying, "*You could like, see actually what you were coding and what you were doing. And the tip that you gave us to, like, make an icon show to know that everything's working. That was fun. I liked it.*"

Yet, a few students felt troubleshooting was sometimes an easier process with screen-based resources than with physical ones. When debugging, P4 not only found it easier to identify which parts of a code she desired to change in on-screen environments, but stated that physical computing makes it harder to pinpoint the source of errors. She said, "*With the physical computing, you know how you want it to go and you can see immediate results. But sometimes you'll think that you have the code correct. And it's not, and you can't figure it out. But with the online coding, you see that it's not working, you know, it's not working. And I can easily highlight where I want to change.*" P13 made a similar point during a focus group where she claimed when working in the web-based StarLogo Nova that errors in the code would always originate from the user as opposed to in physical computing, where the hardware would also be suspect: "*I like in StarLogo Nova how if something's not working... I know that there's something in the code that I need to change or fix. Whereas with the Makeblock, it's kind of like, 'Is this a me thing, or is this... a hardware thing?'*"

5.2.2 Troubleshooting by Coding Modality. Just as the output and feedback of physical and screen-based resources affected how students recognized issues, the type of coding used in a project impacted how students approached troubleshooting. While text-based coding often posed more of a troubleshooting challenge for students, some also found the exercise more beneficial in terms of creative freedom and learning through one's mistakes. P1 reflected on how text-based coding and its more involved troubleshooting process allowed her to explore coding, discover how to code, and assign meaning to code. She said, "*I like the style of the Makeblock coding more. I didn't really like the block-coding as much. I felt like the Makeblock code—and also with our first unit with the music—it was a lot more freeing... It was a lot of trial and error. And [I] could really figure out what I still wasn't sure about with how to code stuff and what stuff meant.*"

In comparison, students approached block-based coding with relative ease, with troubleshooting primarily focusing on undesired outcomes, rather than coding errors. Many students shared the sentiment that they found block-based coding to be fun; however, they also recognized its limitations when compared to text-based coding, acknowledging the creative and learning possibilities that P1 spoke of. As P6 put it, "*I'd want to do more block coding but I know that I also have to learn, you know, other sophisticated coding types, I guess. But I did want to, like improve. So even though block coding is fun, and I would like to use it, I would still want to learn more about different types.*"

We also noted that regardless of whether a student enjoyed text-based coding, they valued the benefits offered by the text editor. During one focus group, several students agreed that they preferred writing code on Jupyter Notebooks over writing code on paper, due to the visual, color-coded feedback of the editor.

> **Interviewer:** "You all agreed that the Jupyter Notebooks and getting to work on a code was helpful. What do you think about when we wouldn't be on

> the computer working on code, but we would be writing it out like on the worksheets? How was that in comparison?"
> **P14:** "Much more difficult."
> **Interviewer:** "More difficult?"
> **P14:** "Because I have to recall from memory and it's weird, because it doesn't show us in color."
> **P7:** "Yeah, that's true. I like the color; makes me know I'm doing it right."
> **P3:** "And you can't always tell like, if you made a mistake with like, if you spelled something wrong, it doesn't tell you necessarily like you kind of you have to pay more attention."

These students responded positively to the comparatively minimal visual output that the Jupyter Notebook interface provided. They cite the editor's error checking and auto-formatting, both of which visually change the coding environment, as the reason they would rather use the interface than hand-write code. They notably do not mention the text-based console feedback that is also lost when only using paper.

6 DISCUSSION

As we discuss our findings, we want to revisit the definitions given in Section 1.1. We observed instances where student learning was primarily subject to an activity's coding modality—the means by which code is written—and its output modality—the means by which output is given. Feedback was information derived by an individual from output, which we saw had direct and indirect implications on troubleshooting processes. In short, our findings highlighted how coding and output modality affected how students recognized feedback (RQ1a) and engaged in troubleshooting (RQ1b). In this section, we discuss these results by 1) highlighting the notable differences between output and coding modalities, and 2) exploring the implications of teaching methods with each modality.

6.1 Differences in Coding and Output Modalities

6.1.1 Output & Feedback. The findings illustrated how the meaning, or lack of meaning, that students assigned to feedback varied by the output modality, with the threshold of meaning dependent upon the individual. The feedback regarded as the least meaningful often came from the text-based output, which was the primary form of output for screen-based computing activities completed in BUS. Some students found the feedback these outputs provided less helpful than outputs from other modalities (P3 and P4), while others dismissed it entirely (P2).

Most students found the feedback from outputs with very sensory experiences the most meaningful. This is in line with many prior works that cite tangible outputs as the reason for higher motivation and yielding natural connections for learners [12, 21]. Students generally reacted positively to the tangible nature of Makeblock Arduino, as well as the musical and visual feedback of EarSketch. Because students who generally disliked screen-based resources (e.g., P2) enjoyed working with EarSketch, this suggests that it is the modality of the output, rather than input, that impacts how meaningful the feedback is.

While students did not regard the text output modality as highly meaningful, we are not proposing that encouraging students to find meaning in text output is futile. Rather, students struggled to know how to look for feedback when it came to screen-based computing with text output. P7 referred to text output as "*random print statement[s] or number letter string[s],*" showing how she viewed the feedback from this output as nearly indecipherable. Compared to what is known about how programming error messages are interpreted [4, 5, 8, 33], it is surprising to see that even user-controlled text output is sometimes difficult to draw meaning from. And even when students found meaning in text-based output, it was seen as underwhelming and unfulfilling. This displays a clear need to improve the recognition of text-based feedback as important and meaningful.

6.1.2 Troubleshooting. When comparing troubleshooting in screen-based modalities to physical modalities, many students expressed a strong preference for physical resources, finding that the tangible output made pinpointing errors in code much clearer. Additionally, students expressed that the rapid prototyping process afforded by physical computing allowed for incremental ideation and troubleshooting, catching errors easily and often. While this incremental process can be (and is frequently) employed in a screen-based setting, students such as P8 and P9 indicated a lack of awareness of the extent to which troubleshooting methods can be applied.

P4 shared the general sentiment that "seeing" feedback was easier with physical computing and, therefore, helpful to troubleshooting, but made an important distinction between knowing when she's made a mistake and knowing how to pinpoint an error. With physical output modalities, she felt that she could immediately see an error, but could not necessarily pinpoint its cause, especially if she had thought that her code was correct. She surmised that screen-based resources made it easier for her to troubleshoot than physical resources. This is an example of how affordances for troubleshooting are dictated by the type of output modality and the manner in which students interpret them.

P1 drew a similar conclusion to P4, but did so by comparing text-based coding and block-based coding in terms of troubleshooting. P1—one of the few students to focus on coding modality rather than output modality—attributed her preference for text-based coding to the incremental troubleshooting process she established, which she felt encouraged her to learn coding on a deeper level. While P1 is unique in voicing this idea, other students also assumed there was greater learning potential for text-based coding. They referenced the limitations of block-based coding, as covered by DiSalvo [10] and Weintrop and Wilensky [35]. However, those same students did not seem able to specify the potential benefits of text-based coding, such as the incremental writing and testing that P1 recounted. It is notable that P1 had comparatively higher levels of experience with computer programming, which may be correlated to her thoughts on text-based coding. Regardless, this awareness of a greater learning potential shows that transitions between coding modalities deserves more attention, particularly when it comes to troubleshooting.

6.2 Recommended Approaches to Computing Education Activities

Throughout the course, we found generally higher acceptance and enthusiasm towards physical computing activities, even though students championed activities in all modalities. We also found a significant number of students who drew issues with text output across all resources. However, the purpose of this research was not only to determine how each modality was experienced by students but to also determine how affordances found in one modality can inform further approaches to others (RQ2). By further reflecting on our findings, it became clear that the range of feedback gained and troubleshooting methods taken were vast and under-explored. Here, we provide a list of recommendations for approaches to CS ed activities based on further developing these facets of computing education.

1. Enable a range of expressive output to be available in screen-based resources. Our efforts to create an engaging and immersive experience in various computing subjects could not appeal to everyone at all times. For example, in the AI unit, some students had issues with the screen-based, text-coding, text-output activities. When reflecting on differences in the units, P4 inspired us to look toward incorporating artistic expression with AI to improve the AI unit output and feedback, without fundamentally changing its content. She mentioned that "*creating an AI that could make art,*" a concept that is entirely feasible, would have made her more excited to explore AI. Therefore, screen-based resources should be designed with a range of potential expressive outputs in addition to textual feedback. Findings in Love and Asempapa [18] reinforce the idea that expressive screen-based CS activities (e.g., Scratch), compared to physical computing activities, can create a more positive disposition towards coding in the classroom and applicability in their career and future. Enabling more expressive output in screen-based resources could increase user engagement and positively affect students' perceptions of coding, while preserving the activity as screen-based.

2. Teach students how to interpret output. We taught students how to use CS materials and solve CS programming problems by focusing on methods and processes. This instruction presented output as a form of success that a program was operating but stopped short of teaching students how to interpret and apply this output. As a result, students like P2 and P7 did not recognize the significance of basic forms of output, and most of the class preferred tangible or visual output to indicate success. Since a large amount of research on student perceptions of feedback while coding focuses on programming error messages [5, 8], we see room to expand this field to perceptions of feedback when errors are not present to ensure learning is still happening to the fullest degree. CS educators should increase guidance in the interpretation of output so that students can identify meaningful feedback from all forms of output.

3. Treat some forms of output as midway points to a final goal. Some students who recognized meaningful feedback from text output did so with low interest and enthusiasm. For example, text output assisted some students in debugging their code; however, students did not feel text output alone was a satisfying final result of a project. Rather than avoiding exposing students to text output, educators could situate text output as an incremental step toward a finished product that employs other modes of output for the final steps. We anticipate that if unpopular forms of output are not seen as the culmination of a student's work, but as an important step in the process, it will destigmatize that output from being considered unfulfilling. This interpretation of student experiences suggests that many of the cited benefits of physical computing (e.g., [12, 18, 21]) may be due to students' satisfaction with the finality and completion of tangible output opposed to an intermediary message from text output.

4. Teach how to troubleshoot in all coding and feedback modalities. In each modality, whether physical or screen-based output or text-based or block-based coding, some students spoke about an inability to test their code before it was finished. While one can incrementally test their code in all of these modalities, students did not always feel capable of doing so. Computer scientists would agree that rapid prototyping solutions through trial-and-error is common—if not necessary—in modern computing. Accomplishing this iterative process also takes less time using screen-based resources than physical resources. However, students did not see this advantage, resulting in students seeing troubleshooting with physical resources as superior. This suggests an increased emphasis on teaching students how to apply an incremental approach to coding will provide students with a more effective model for troubleshooting. The effectiveness of the incremental coding approach could be seen when we emphasized that students update the LED matrix icon on their micro:bits when uploading new code. P8 spoke about appreciating this lesson and we observed that other students found it useful, which suggests it helped them reach the learning objective. Following this approach will enable students to understand their code on a deeper level and incorporate troubleshooting fluidly into their problem-solving process.

6.3 Limitations

There are a few limitations of this study that must be considered. While the first author has taught in many informal CS workshops, he did not have formal educational training, and the course itself was not designed to adhere to any formal standards. Additionally, the sampling frame from which this study was conducted was limited by the sponsor's participation requirements and may affect how generalizable the findings are. Students self-selected to participate in the study, which may have influenced how they responded to questions about the program and their experiences.

7 CONCLUSION

Through this study, we developed an understanding of the affordances and drawbacks of popular coding and feedback modalities. Across several CS resources, we saw that the modality did not hinder overall learning, but that students experienced key steps of recognizing feedback and troubleshooting in vastly different ways. As stated by Love [17], research on how physical computing education affects the attitudes of students learning with it is limited, despite the many studies that cite its benefits. This holds true for other forms of computing, as well. By analyzing what students enjoyed, disliked, and reflected upon in the BridgeUP STEM programming course, we provided suggestions on select areas of

teaching CS that break from traditional approaches. Our findings are unique in that few studies of comparative work have been done across resources with varying output and coding modalities.

Coding modality, output modality, feedback and troubleshooting should be treated as four different considerations when looking at what students learn from computing activities. With continued research in this area, we can highlight opportunities for improving screen-based computing based on the affordances of physical computing revealed by the students, and vice versa.

ACKNOWLEDGMENTS

We would like to thank NCWIT for their help and support during this project. We would also like to thank the Hopper-Dean Foundation and the Alfred P. Sloan Foundation for their support. BridgeUP STEM was funded by Helen Gurley Brown Foundation and hosted at Georgia Tech.

REFERENCES

[1] Jonny Austin, Howard Baker, Thomas Ball, James Devine, Joe Finney, Peli De Halleux, Steve Hodges, Michał Moskal, and Gareth Stockdale. 2020. The BBC Micro:Bit: From the U.K. to the World. *Commun. ACM* 63, 3 (feb 2020), 62–69. https://doi.org/10.1145/3368856

[2] Thomas Ball, Abhijith Chatra, Peli de Halleux, Steve Hodges, Michał Moskal, and Jacqueline Russell. 2019. Microsoft makecode: embedded programming for education, in blocks and typescript. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E*. 7–12.

[3] Massimo Banzi and Michael Shiloh. 2022. *Getting started with Arduino*. Maker Media, Inc.

[4] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do developers read compiler error messages?. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 575–585.

[5] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE-WGR '19)*. Association for Computing Machinery, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[6] Virginia Braun and Victoria Clarke. 2012. Thematic Analysis. *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological*. 2 (2012), 57–71. https://doi.org/10.1037/13620-004

[7] Renata Burbaitė, Robertas Damaševičius, and Vytautas Štuikys. 2013. Using robots as learning objects for teaching computer science. In *X world conference on computers in education*. 101–110.

[8] Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B. Powell. 2021. On Designing Programming Error Messages for Novices: Readability and Its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 55, 15 pages. https://doi.org/10.1145/3411764.3445696

[9] Kayla DesPortes and Betsy DiSalvo. 2017. Where are the Glass-Boxes? Examining the Spectrum of Modularity in Physical Computing Hardware Tools. In *Proceedings of the 2017 Conference on Interaction Design and Children*. 292–297.

[10] Betsy DiSalvo. 2014. Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science. *IEEE computer graphics and applications* 34, 6 (2014), 12–15.

[11] Shuchi Grover, Roy Pea, and Stephen Cooper. 2015. Designing for deeper learning in a blended computer science course for middle school students. *Computer science education* 25, 2 (2015), 199–237.

[12] Steve Hodges, Sue Sentance, Joe Finney, and Thomas Ball. 2020. Physical computing: A key element of modern computer science education. *Computer* 53, 4 (2020), 20–30.

[13] IBM. 2021. IBM SkillsBuild - AI Foundations: A Collaboration of ISTE and IBM. https://students.yourlearning.ibm.com/activity/PLAN-B2125F145F0E?channelId=CNL_LCB_1596575854335

[14] Filiz Kalelioglu and Sue Sentance. 2020. Teaching with physical computing in school: the case of the micro: bit. *Education and Information Technologies* 25, 4 (2020), 2577–2603.

[15] Majeed Kazemitabaar, Viktar Chyhir, David Weintrop, and Tovi Grossman. 2023. Scaffolding Progress: How Structured Editors Shape Novice Errors When Transitioning from Blocks to Text. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 556–562.

[16] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). IOS Press, 87 – 90.

[17] Tyler S Love. 2023. Examining middle school students' attitudes toward computing after participating in a physical computing unit. *Interactive Learning Environments* (2023), 1–20.

[18] Tyler S. Love and Reuben S. Asempapa. 2022. A screen-based or physical computing unit? Examining secondary students' attitudes toward coding. *International Journal of Child-Computer Interaction* 34 (2022), 100543. https://doi.org/10.1016/j.ijcci.2022.100543

[19] Brian Magerko, Jason Freeman, Tom Mcklin, Mike Reilly, Elise Livingston, Scott Mccoid, and Andrea Crews-Brown. 2016. EarSketch: A STEAM-Based Approach for Underrepresented Populations in High School Computer Science Education. *ACM Transactions on Computing Education (TOCE)* 16, 4 (2016), 1–25. https://doi.org/10.1145/2886418

[20] Makeblock. 2022. Makeblock. https://www.makeblock.com/

[21] Paul Marshall. 2007. Do Tangible Interfaces Enhance Learning?. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (Baton Rouge, Louisiana) *(TEI '07)*. Association for Computing Machinery, New York, NY, USA, 163–170. https://doi.org/10.1145/1226969.1227004

[22] Sandra Y Okita. 2014. The relative merits of transparency: Investigating situations that support the use of robotics in developing student learning adaptability across virtual and physical computing platforms. *British Journal of Educational Technology* 45, 5 (2014), 844–862.

[23] Jin Pan. 2016. *Performance Engineering of the StarLogo Nova Execution Engine*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[24] Mareen Przybylla and Ralf Romeike. [n. d.]. Key Competences with Physical Computing. *KEYCIT 2014* ([n. d.]), 351.

[25] Aaron Rasheed Rababaah and Ahmad A Rabaa'i. 2018. Enhancing programming learning environment with physical computing and robotics: a case study of the American University of Kuwait. *International Journal of Teaching and Case Studies* 9, 4 (2018), 323–346.

[26] Mitchel Resnick. 2008. Sowing the seeds for a more creative society. *Learning & Leading with Technology* 35, 4 (2008), 18–22.

[27] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. Design principles for tools to support creative thinking. (2005).

[28] Gabriela T Richard and Sagun Giri. 2019. Digital and physical fabrication as multimodal learning: Understanding youth computational thinking when making integrated systems through bidirectionally responsive design. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–35.

[29] Raja Ridgway. 2018. Project GUTS. *Science Scope* 42, 3 (2018), 28–33.

[30] Sue Sentance, Jane Waite, Lucy Yeomans, and Emily MacLeod. 2017. Teaching with Physical Computing Devices: The BBC Micro:Bit Initiative. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (Nijmegen, Netherlands) *(WiPSCE '17)*. Association for Computing Machinery, New York, NY, USA, 87–96. https://doi.org/10.1145/3137065.3137083

[31] Ben Shneiderman, Catherine Plaisant, Maxine S Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. 2016. *Designing the user interface: strategies for effective human-computer interaction*. Pearson.

[32] Jane Waite. 2017. Pedagogy in teaching computer science in schools: A literature review. *London: Royal Society* 253 (2017).

[33] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2012. Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. In *Advances in Web-Based Learning-ICWL 2012: 11th International Conference, Sinaia, Romania, September 2-4, 2012. Proceedings 11*. Springer, 228–239.

[34] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. 2016. Defining computational thinking for mathematics and science classrooms. *Journal of science education and technology* 25, 1 (2016), 127–147.

[35] David Weintrop and Uri Wilensky. 2015. To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th international conference on interaction design and children*. 199–208.

[36] David Weintrop and Uri Wilensky. 2017. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1–25.

[37] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

[38] Yahboom. 2022. Yahboom micro:bit Robot Car. http://www.yahboom.net/study/Bitbot

A APPENDIX

A.1 BridgeUP STEM Syllabus

Table 2: Syllabus and learning objectives for the BridgeUP STEM Cohort 1A programming course.

Unit 1: EarSketch	
Week 1	Introductions EarSketch Lesson 1 • creating Python scripts • coding music into a DAW • debugging your code EarSketch Lesson 2 • changing tempo • code comments • uploading your own sounds
Week 2	EarSketch Lesson 3 • variables • making beats Genres in music, creating genre songs
Week 3	EarSketch Lesson 4 • for-loops • print statements • control flow EarSketch Lesson 5 • effects • functions • more debugging tips Introduce Final Project: Rescore Videos
Week 4	Finish Rescore Videos Viewing party of Final Mix
Unit 2: Artificial Intelligence	
Week 5	Introduction to AI Types of AI Machine Learning • Components of ML • Types of ML • Supervised and unsupervised learning • Activity: How Do Machines Learn?
Week 6	Python: Back to Basics • Jupyter Notebooks • Algorithms
Week 7	More Machine Learning • Demo: building a neural network • Making a decision tree • Google's Teachable Machine Logic Practice

Week 8	Algorithmic Bias in AI • Game: Survival of the Best Fit AI and Humanity • Risks and Benefits Conditions and Logic • Practice • If-statement review • Decision tree review
Week 9	Ethics in AI Introduce Final Project: Trailblazer Isolation
Week 10	Coding Practice Minimax Algorithm Classes Game-Playing AI • Minimax Player • Game-playing strategies
Week 11	Game-Playing AI • Trailblazer Isolation overview • Project work time
Week 12	Game-Playing AI • Project work time Tournament: Trailblazer Isolation Championship
Unit 3: Arduino Makeblock Kits	
Week 13	Introduction to Makeblock Kits • Parts overview • Arduino IDE overview Sensors
Week 14	Actuators Combining Sensors and Actuators
Week 15	Final Project: Artifact Builds
Unit 4: Complex Adaptive Systems	
Week 16	Introduction to Complex Adaptive Systems • Complex vs. Complicated? • Activity: Turn & Walk StarLogo Nova Overview • Turn & Walk simulation
Week 17	SLN Turtle Activities • Flower Turtles • Trailblazer Turtles • Bumper Turtles
Week 18	Simulation vs. Real-World SLN Turtle Activities • Pandemic models • Measurements • Experimental Design Form Activity: Bug Code

Week 19	Python Review Introduction to micro:bit • Programming the micro:bit • Line-following robots
Week 20	micro-City • Constructions • Robot Parade