



The impact of a block-based visual programming curriculum: Untangling coding skills and computational thinking

Zhanxia Yang^{a,*}, Jessica Blake-West^b, Dandan Yang^b, Marina Bers^b

^a Department of Education, Ithaca College, USA

^b DevTech Research Group, Boston College, USA

ARTICLE INFO

Keywords:

Computer science
ScratchJr
Early elementary education
Computational thinking
Evidence-based curriculum

ABSTRACT

Background: and Context: Computer programming is increasingly being taught to young children worldwide. Quality programming curricula are needed to enable students to explore computer science concepts and develop a positive attitude towards programming. However, few studies have been conducted on the available curricula for young learners.

Objective: This study aims to provide an evidence-based, public, and free computer science curriculum for early childhood education by investigating the efficacy of the CAL-ScratchJr curriculum, among second graders.

Method: A cluster-randomized controlled trial involving 11 schools and 21 second-grade classrooms was conducted to examine the impact of the 24-lesson curriculum on students' coding proficiency and computational thinking.

Findings: The results indicated that the CAL-ScratchJr curriculum intervention had a medium positive effect on second graders' coding skills, but no significant impact on computational thinking.

Implications: Results caution against incorporating disconnected abstract computational thinking practices in education. The features of the curriculum were discussed.

1. Introduction

Programming is an essential skill for the 21st century. It provides a way for people to control the interaction with computers and solve problems via computers in the automated world. The last two decades have seen a growing emphasis on the teaching of programming skills to young children (Bers, 2020; Century et al., 2020; Falloon, 2016; Flórez et al., 2017; Lye & Koh, 2014). Early childhood is an optimal time to introduce computer science, as young minds are particularly receptive to learning and developing new skills. Moreover, it is a time when children are being introduced to other systems of symbolic representation, such as natural language.

Early exposure to programming has many benefits for young children, developing important problem-solving skills, encouraging persistence, and improving digital literacy; computational thinking; creativity; confidence; communication skills; planning and inhibition skills; and attitudes toward computer science (Arfe et al., 2020; Bers, 2008a, 2020, 2021a; Bers et al., 2014; Brooks & Phillips, 2017; Bruckman et al., 2002; Burke, 2012; Burke & Kafai, 2014; Relkin et al., 2020, 2021). Additionally, early exposure to programming can foster young

learners' interest in technology and engineering and reduce or even preclude stereotype threat (Sullivan & Bers, 2013; Yang & Bers, 2023).

Given the various benefits of instruction in programming, evidence-based early elementary education in computer science and programming is in high demand (Manches & Plowman, 2017). This paper aims to provide evidence of how a block-based visual programming curriculum, the Coding as Another Language-ScratchJr curriculum (the CAL-ScratchJr curriculum), has impacted second-grade students' coding skills and computational thinking with a cluster randomized control trial design.

1.1. Early childhood CS curriculum

As more and more attention has been drawn to teaching and learning computer science (CS) and programming in early childhood education, educators, researchers, and educational technology designers have started to introduce CS and programming curricula through a variety of activities such as visual block-based programming, simple coding games, robotics, and interactive storytelling (Bers, 2008b, 2019; Bers & Cassell, 1999; Code.org, CSTA & ECEP Alliance, 2022; Kalelioglu, 2015;

* Corresponding author.

E-mail addresses: jyang1@ithaca.edu (Z. Yang), jess.blake-west@bc.edu (J. Blake-West), yangbno@bc.edu (D. Yang), bers@bc.edu (M. Bers).

<https://doi.org/10.1016/j.learninstruc.2024.102041>

Received 18 April 2023; Received in revised form 25 July 2024; Accepted 20 October 2024

0959-4752/© 2024 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

Saez-Lopez et al., 2016). Designed to be engaging and fun, many of these curricula have been demonstrated to be developmentally appropriate for young learners. Table 1 summarizes key features of the most frequently used programming curricula designed for the early childhood age group.

These curricula show many similarities: for example, they all address the most common programming concepts, they are all free to a certain extent, and they all focus on the elementary levels. Some differences are evident, such as the user interface of the platforms, while others are less noticeable, such as pedagogical approaches built into the design of the respective software used by the curricula (see Table 1). For example, with codeSpark Academy and Code.org curricula, students can solve puzzles (or finish games) with predefined solutions or destinations as they program using the given software. Because students' coding is guided by specific questions which have predefined correct answers, an error message is returned if the coding is not following the predefined solutions; by contrast, ScratchJr and the activities presented in the CAL-ScratchJr curriculum are open-ended and flexible (Bers et al., 2023; Blake-West & Bers, 2023).

In addition to the differences in pedagogical approach among the three curricula, the unplugged activities are different too. The unplugged activities in Code.org are based on explicit and abstract computational thinking practices, such as pattern recognition and decomposition on a worksheet. In comparison, the unplugged activities in codeSpark Academy and CAL-ScratchJr are hands-on activities, for example, the "program the teacher" activity in the CAL-ScratchJr curriculum, in which students "program" the teacher by giving detailed instructions to the teacher to perform a task, or the "Splash Clash" outdoor activity in codeSpark Academy.

Furthermore, the CAL-ScratchJr curriculum offers the option to customize characters and scenes, aligning with its emphasis on personally meaningful projects. By contrast, Code.org and codeSpark Academy do not provide this feature. In codeSpark Academy, after the guided games, children can later create their own games, but they must use given characters and cannot create new characters. Grade level coverage is another difference: Code.org and the CAL-ScratchJr curriculum have grade-level specific curricula for each grade, while the grade in codeSpark Academy is not specified. Of the three, the CAL-ScratchJr curriculum is the only one that specifically focuses on early childhood learners by presenting a coding playground (Bers, 2019; Bers et al., 2023).

The CAL-ScratchJr curriculum is unique in that it advocates for coding as a form of expression paralleled with natural language and

promotes creativity through the design of personally meaningful projects. Just as letters and words are a system of representation, the blocks in ScratchJr can be combined to convey different meanings or events (Bers, 2019; Unahalekhaka & Bers, 2022a). This parallel is enforced throughout the curriculum in multiple ways. For example, the first activity and discussion of the CAL-ScratchJr curriculum are centered entirely around understanding the concept of languages, before even introducing ScratchJr.

The CAL-ScratchJr curriculum also utilizes the parallels between coding and natural language to support algorithmic thinking or sequencing (Bers et al., 2023; Blake-West & Bers, 2023). This reinforcement is realized both in simple ways, such as an "order matters" activity in which children scramble words in a sentence, and in complex ways; for example, centering the two largest coding projects around storytelling, in which identifying a beginning, middle and end of the story are crucial to the outcome of the story, just as the order of a program can alter the outcome. ScratchJr reflects the coding-literacy connection in other ways as well; for example, the direction in which blocks are put together (Blake-West & Bers, 2023). Unlike typical programming languages, which give commands top down, ScratchJr syntax is entirely Left to Right, to mirror the direction of letters and words in natural languages such as English. Additionally, ScratchJr is designed for story telling—a main feature of the app is the ability to add and connect multiple pages with different programs, so that a story with multiple scenes can be created.

Bers has described the different types of CS curricula as promoting either a "coding playground" or a "coding playpen" (Bers, 2012; Bers, 2014; Bers, 2020; Bers, 2022; Bers et al., 2019). The CAL-ScratchJr curriculum, designed to foster the former, draws inspiration from Constructionism (Papert, 1980). It aims to provide opportunities for students to actively explore and create; ensuring the learning is personally meaningful (Ausubel, 1963; Mayer, 2002). The open-endedness feature and the design-based approach provide opportunities for such meaningful learning in the context of a coding playground: instead of solving problems with a predefined path, as in the codeSpark Academy and Code.org curricula, students create their own animated projects through the CAL-ScratchJr curriculum.

1.2. Previous studies examining impact of CS curriculum in early childhood

CS curricula for young learners have limited peer-reviewed

Table 1
Key features of three programming curricula.

Features	CAL-ScratchJr Curriculum	codeSpark Academy Curriculum	Code.org curriculum
Cost	Free	Free for at-school use Not free for at-home use	Free
Included Grade levels	K-2	All elementary	K-5
Grade-level specific	Yes	No	Yes
Main CS Concept	Sequencing, Algorithms, Parameters, Loops, Events, Conditionals, Parallel programming, Debugging, Design Process	Sequences, Algorithms, Parameters, Loops, Events, Conditionals, Stacks and Queues, Debugging, Decomposition	Sequencing, Algorithms, Debugging, Loops, Conditionals, Functions, Variables, Events, Data
Pedagogical Approach	Open-ended problem solving and opportunities for content creation	Puzzles with specific solutions; Option of game creation at the end	Puzzles with specific solutions
Interface of Technology/Platform	Puzzle-shaped word-free programmable blocks with simple hover words for pre readers (left to right); Option of creating characters and customizing the existing ones; Option of creating scenes and customizing the existing ones	Puzzle-shaped word-free programmable blocks (left to right); Narrator reads the question stems; Premade characters for the given games; Premade scenes	Moderate text-based programmable blocks (top down); Optional narrator for question stems; Premade characters for the given problems; Premade scenes
User's focus throughout curriculum	Create animations; Story-telling	Solve puzzles from predefined solution(s)	Solve puzzles from predefined solution(s)

intervention studies, conducted primarily as after-school activities or with selected participants (Lye & Koh, 2014). The studies generally report improvement on students' programming skills and computational thinking over time. However, the absence of a control group makes it difficult to determine whether the improvement was due to the interventions, or to other factors like familiarity with the test or maturation (Denniston et al., 2015; Lahullier, 2019). A few experimental studies reported mixed results of Code.org, CAL-ScratchJr, codeSpark Academy, CAL-KIBO robotics, and Scratch curricula conducted in various educational settings.

Two experimental studies have examined the impact of Code.org on computational thinking and coding performance in young learners. Arfe et al. (2020) found significant improvements in coding skills among first graders in Italy, and Oluk and Çakir (2021) reported enhanced computational thinking among sixth graders in Turkey. While these results are promising, there are areas that could benefit from further exploration. For instance, both studies provide limited details on group formation, classroom implementation, and teacher involvement. Additionally, the interventions were relatively brief, lasting within six weeks. Arfe et al. (2020) reported gender differences in their baseline, which limited the generalizability of the findings. Oluk and Çakir (2021) broadly defined computational thinking to include creativity, critical thinking, problem-solving, and algorithmic thinking, but did not detail the criteria for their quasi-experiment. Given that these studies were conducted in Italy and Turkey, more experimental research across various cultural and educational contexts, particularly in the U.S. (Lahullier, 2019), is necessary to validate the generalizability of these findings.

In addition to the effect of Code.org curriculum, Relkin and co-authors examined the effect of multiple CS curricula intervention effects on kindergarten to second-grade students' computational thinking and reported mixed results. Specifically, the interventions of CAL-ScratchJr and codeSpark Academy on students' computational thinking were not statistically significant, while the intervention of CAL-KIBO robotics curriculum showed positive effect compared to the control group when controlling for the baseline and clustering effect of classrooms, $p < 0.001$, $CI = [0.78, 2.23]$ (Relkin, 2022). Similarly, Relkin et al. (2021) conducted a quasi-experiment among first and second graders to examine the impact of the intervention of CAL-KIBO robotics curriculum on computational thinking and found a significant effect among first graders (Bayes factor > 100) but not second graders (Bayes factor = 1.88). However, Relkin et al. (2021) reported that significantly higher baseline computational thinking scores were found in the CAL-KIBO robotics curriculum intervention group, which could be a confounding reason for the significant improvement in the intervention group. More experimental studies with comparable baseline of computational thinking are needed to verify programming curricula impact on young students' computational thinking.

Likewise, Grillo-Hill et al. (2019) examined the impact of codeSpark Academy and did not find a difference between the intervention and control group in the domain of computational thinking—sequencing. Straw et al. (2017) found that the Code Club intervention positively impacted students' coding skills but showed no corresponding improvement in their computational thinking skills compared to a control group. Similarly, Woo and Falloon (2022) reported that computational thinking was not an associated product of coding if students did not practice the cognitively demanding coding skills.

A study by Laurent et al. (2022) explored the impact of Scratch (for children ages 8–16) programming intervention on the mathematics learning of fourth and fifth graders from 46 randomly assigned schools. Their findings revealed small but significant negative effects on the three mathematics concepts investigated (Euclidean Division, Additive Decomposition, and Fractions). Specifically, the intervention group exhibited a significantly lower performance on all three concepts, with effect sizes of -0.16 , -0.19 , and -0.21 , respectively, compared to their counterparts in the control group. These results indicate that students who received Scratch programming instruction did not perform as well

on the assessed mathematics concepts as those who did not. Laurent et al. (2022) concluded that transfer of learning between programming and mathematics domains may not occur automatically and emphasized the importance of examining such transfer across different age groups. More studies are needed to verify the effect of CS curricula on young learners' cognitive measures such as computational thinking. The current study is an experimental study examining the effect of CAL-ScratchJr curriculum in real second grade U.S. classrooms, contributing to this broader body of research.

1.3. Coding and computational thinking

The terms *computational thinking skills* and *coding skills* are sometimes used interchangeably; however, they are different concepts (Zhang & Nouri, 2019). *Coding skills* refers to the ability to understand and generate programming languages, while *computational thinking skills* designates a broader set of problem-solving skills applicable to various aspects of life and professions (DePryck, 2016; Wing, 2011; Yadav et al., 2017). Just as proficiency in natural languages facilitates communication and expression, being able to code in a programming language allows the creation of personally meaningful projects (Bers, 2019). The cognitive and metacognitive processes involved are known as computational thinking (Barr & Stephenson, 2011; Grover & Pea, 2013; Wing, 2006).

Wing (2006) defined *computational thinking* as a fundamental skill set which can promote problem solving and can sometimes be associated with programming. Cuny et al. (2010) defined *computational thinking* as a thought process involved in formulating problems and solutions that can be efficiently carried out by an information processing agent. While Wing (2006, 2011) defined *computational thinking* as a thought process, Papert (1980, 1996) emphasized the interactive programming experience as the primary vehicle for developing such computational thinking. Bers further expanded on this topic, noting that the thought processes involved in computational thinking must be expressed through a language, and that programming languages are well-suited to this purpose (Bers, 2021b).

Recently, the emphasis in computer science education has shifted from teaching only programming to include development of computational thinking (Fayer et al., 2017). Research has shown that integration of computational thinking skills into the academic curriculum can enhance children's problem-solving abilities (Barr & Stephenson, 2011; Bers, 2021b; Chen et al., 2017; Papert, 1980; Wing, 2011; Yadav et al., 2017), critical thinking skills (Kirschner et al., 2004), and creativity (Bers, 2021b). It can also help develop children's ability to think logically and systematically, which are essential skills for both their academic and personal development (Wing, 2006).

Although computational thinking is recognized as an important skill related to coding performance, experimental studies in early computer science classrooms have not consistently supported this claim (Grillo-Hill et al., 2019; Oluk & Çakir, 2021; Relkin, 2022; Relkin et al., 2021; Straw et al., 2017). These studies used various measurements for computational thinking, adding to the challenge. To verify previous findings, this study examines the effects of the CAL-ScratchJr curriculum on both computational thinking and coding performance among second-grade students.

1.4. The problem statement and study purpose

Despite the increasing importance of computer science and programming education, the options for early childhood curricula are few; among these, even fewer that are evidence-based and publicly available at no cost. Previous research on the impact of computer science curricula for early education has been limited, and those studies that have been conducted yielded mixed results regarding students' computational thinking. This study is intended to fill the gap by evaluating the effectiveness of the CAL-ScratchJr curriculum in classroom settings, using

experimental research designs. The purpose of this study is to examine the specific impact of the CAL-ScratchJr curriculum on second-grade students' coding performance in ScratchJr and on their computational thinking skills.

1.5. Research questions

- (1) What is the impact of the CAL-ScratchJr curriculum in second-grade classrooms on the development of students' coding skills?
- (2) What is the impact of the CAL-ScratchJr curriculum in second-grade classrooms on the development of students' computational thinking?

2. Method

2.1. Participants and context

The participating students comprised 331 second-grade students from 11 public schools and 10 school districts on the East Coast of the United States. The students had an average age of 8.4 (SD = 0.4), ranging from 7.1 to 9.9, with a gender distribution of 49.4% female and 50.6% male. In terms of ethnicity, 64.6% were White, 18.3% were Hispanic, 6.1% were African American, 4.9% were Asian, 4.6% identified as two or more races, and 1.2% were American Indian or Alaska Native. Regarding enrollment in special programs, 10% had an Individualized Education Plan (IEP) and 11% had limited English language proficiency (LEP). Regarding students' socio-economic status (SES), 21% were part of the free or reduced lunch program (see Table 2).

2.2. Research design

This study is part of a multi-year project funded by the US Department of Education, examining the impact of an intervention on second graders (see Fig. 1). The current study uses an experimental design, specifically a cluster-randomized control trial study. The 11 schools were randomly assigned to either intervention or control groups. Teachers in the intervention group received professional development training and implemented the CAL-ScratchJr curriculum with their students. In contrast, teachers in the business-as-usual control group did not undergo professional training. Students in their classrooms, assigned to the business-as-usual control group, followed the regular school curriculum rather than the CAL-ScratchJr curriculum employed in this experimental study. Consent forms were obtained from school principals, teachers, and parents of participating students, who were assessed on their coding performance and computational thinking before and after the intervention. The Institutional Review Board approved the study.

2.2.1. Randomization procedure

Schools were randomized within blocks. Specifically, three blocks were formed to ensure comparable demographics and comparable

sample sizes: 1) Schools in the lowest and second-to-lowest poverty quartile that expect fewer than 100 students to participate; 2) schools in the lowest and second-to-lowest poverty quartile that expect more than 100 students to participate; and 3) schools in the highest and second-to-highest poverty quartile. As a result of random assignment at the school level, five schools containing nine classrooms were assigned to the intervention group, and six schools containing 12 classrooms were assigned to the control group.

Participating teachers welcomed and encouraged all students to participate in this research study and distributed the parental consent form (both digital and paper, available in multiple languages) to their students, and a site coordinator collected the forms from the teachers for the research team. Fig. 2 presents information about attrition of this study in both the treatment and the control conditions. The attrition rate is low, and the analyzed samples are approximately 95% of all consented student participants. Appendix A presents a comparison of the characteristics of attrited students between the two conditions. All the research assistant assessors who conducted a one-on-one coding games assessment were blinded to the school assignment.

Through Zoom, an experienced trainer conducted professional development (PD) training to introduce the CAL-ScratchJr curriculum to all teachers in the intervention group. Intervention group teachers received their PD training in August 2021. The PD lasted 4 hours. During the time, teachers explored both the pedagogical and content knowledge of the CAL-ScratchJr curriculum and were introduced to the ScratchJr app via guided play. The PD successfully improved teacher's coding knowledge in ScratchJr and their self-efficacy in teaching coding and ScratchJr to their students (Kapoor et al., 2022). Additionally, designated technology specialists and the research team were available to support teachers throughout the course of curriculum implementation.

At the beginning of 2021/22 school year, the consented students were assessed one-on-one with trained research assistants on their baseline (pre-curriculum) coding performance assessment, as measured by CSA, and their baseline computational thinking, as measured by TechCheck. Students' baseline CSA and TechCheck assessments started in October 2021 and finished in January 2022. Consented students were assessed again on their coding assessment and their computational thinking at the end of school year 2021/22 between May and June 2022.

2.3. Measures

Coding Stages Assessment (CSA). The CSA is a validated tool used to evaluate a child's coding performance in ScratchJr (de Ruiter & Bers, 2021). It consists of 30 questions total covering five stages of coding in ScratchJr. These stages are Emergent, Coding and Decoding, Fluency, New Knowledge, and Purposefulness, in increasing order of proficiency. The stages are based on a theoretical framework that posits the learning of a programming language to be akin to the learning of a written natural language (Bers, 2019). The CSA aims to determine which stage a child is in by asking them six questions from each stage. The questions involve many CS concepts integrated in ScratchJr such as sequencing, events, loops, and parallel programming (See Appendix B for a sample question). This validated assessment has been used in previous studies (Bers et al., 2023; Kapoor et al., 2022; Yang & Bers, 2024; Yang et al., 2023; Unahalekhaka & Bers, 2022b).

Well-trained research assistants administered the assessment via Zoom synchronously by sharing a window with visual prompts for each question in the assessment and narrating each question to participating students. Participating students then coded in the ScratchJr app on their own device and presented their ScratchJr coding via Zoom with verbal explanations (when applicable). The research assistants rated students' responses to each question either as "Satisfactory" or "Unsatisfactory." If a student answered 5 out of 6 questions with a "Satisfactory" rating in one stage, the student proceeded to the next stage; if not, the assessment ended with the stage completed. The student's coding stage is determined by the last stage they passed. The total score is calculated by

Table 2
Demographic information of the participating students.

	N	Percentage
Gender (Female)	164	49.4%
Ethnicity		
White	214	64.6%
Hispanic	61	18.3%
African American	21	6.1%
Asian	16	4.9%
Two or more races	15	4.6%
American Indian or Alaska Native	4	1.2%
Enrollment in Special Education Program		
IEP	33	10%
LEP	36	11%
SES (Free/Reduced lunch)	70	21%

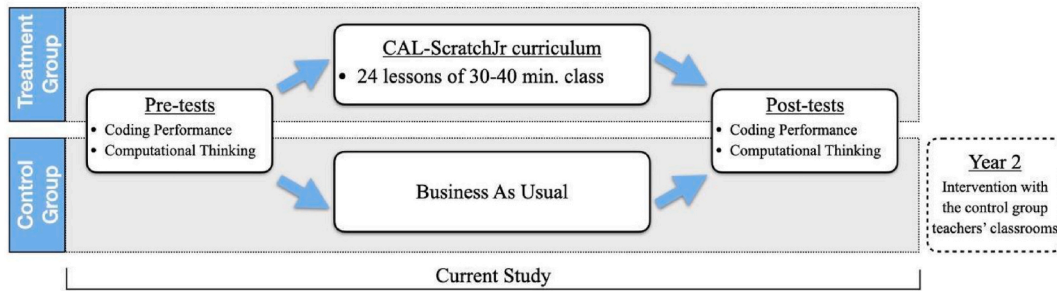


Fig. 1. Study design.

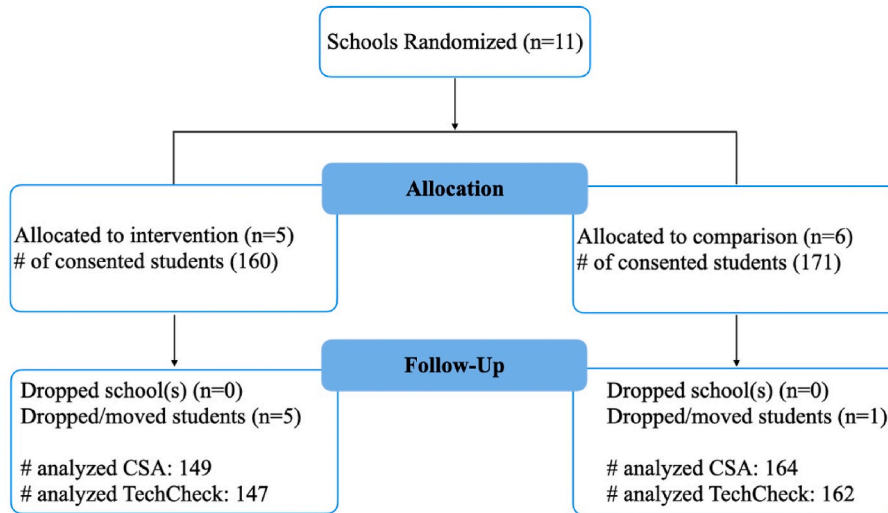


Fig. 2. Attrition diagram.

giving more weight to points earned in higher stages, with a maximum possible score of 39 points (de Ruiter & Bers, 2021).

All research assistants completed CSA training and demonstrated great reliability, achieving a minimum Cohen's Kappa of 0.82 with an experienced rater before administering the CSA. To further ensure inter-rater reliability, about 15% of students were rated simultaneously by a second qualified rater on Zoom. The rater's camera was turned off during the process to minimize potential disruptions. The data from this study indicated substantial interrater reliability, both in the pre-CSA (Cohen's $\kappa = 0.84$) and post-CSA (Cohen's $\kappa = 0.92$) ratings. Furthermore, Guttman's Lambda 6 (split-half reliability) indicated good internal consistency of both pre-CSA ($\lambda_6 = 0.77$) and post-CSA ($\lambda_6 = 0.68$).

TechCheck. The TechCheck assessment is a validated tool used to evaluate young children's computational thinking (Relkin et al., 2020; DevTech Research Group). As such, the TechCheck assessment for second graders includes 15 *unplugged* multiple-choice questions covering six domains in computational thinking: hardware/software, debugging, algorithms, modularity, representation, and control structures, as derived from Powerful Ideas (Bers, 2017; Papert, 1980). The 15 questions employed various tasks to explore these domains, including sequencing challenges, shortest path puzzles, missing symbol series, object decomposition, obstacle mazes, symbol shape puzzles, identifying technological concepts, and symmetry problems (Relkin et al., 2020). Several sample questions are provided in Appendix B. *Unplugged* means that the questions do not require a computer to solve them; there is only one correct choice in each question. Each correct response is worth one point, which makes a total score of 15 points. Two non-computational thinking practice questions (e.g., Which of the following is an animal?) were asked at the beginning to familiarize students with the

assessment format, but were not counted in the total score. Responses to all questions were required to complete the assessment. This validated assessment has been used in previous studies (Relkin, 2022; Relkin & Bers, 2021; Relkin et al., 2021, 2023).

All TechCheck questions were administered one-on-one with trained research assistants via Zoom. The research assistant narrated the question to the students with a shared screen of the questions; the students responded to prompts on the assessment verbally by telling the research assistant the answer choice. The research assistant clicked the responses for the students.

When examining internal consistency, the pre-TechCheck assessment demonstrates a Cronbach's alpha of 0.52, while the post-TechCheck assessment reveals a Cronbach's alpha of 0.63. Although these alpha values may not be considered high, it is important to note that TechCheck encompasses several computational thinking domains guided by prior research, which supports a multi-domain rather than a unified nature of the construct of computational thinking. Due to the inclusion of multiple domains, it is likely that this approach resulted in a relatively lower overall internal consistency when compared to assessments that focus solely on a single domain of computational thinking. However, with its multi-domain structure, TechCheck provides a more comprehensive and holistic measure of computational thinking.

2.4. Intervention

The CAL-ScratchJr curriculum has 24 lessons of 45 min each, which introduce a substantial depth and a wide breadth of blocks and ScratchJr functions to the second graders (Bers, 2019; Bers et al., 2023). Fig. 3 presents a roadmap of the second-grade CAL-ScratchJr curriculum. Each

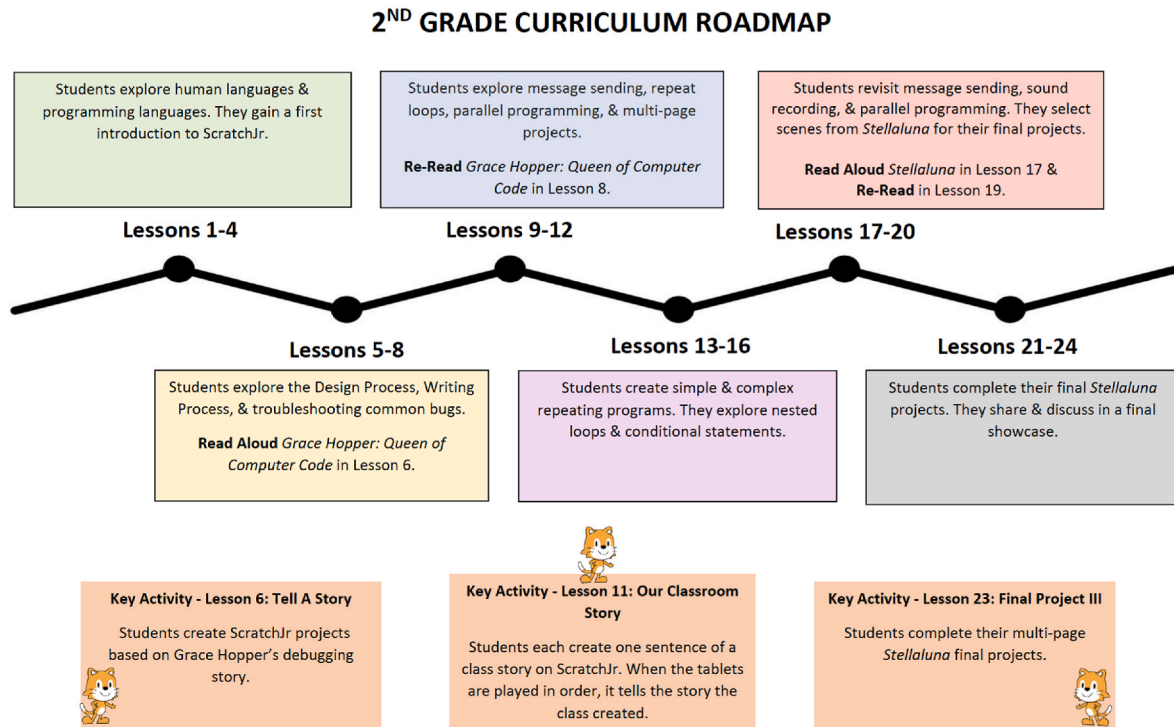


Fig. 3. Second-grade CAL-ScratchJr curriculum roadmap.

lesson is structured by the following activity formats: Warm-Ups; Opening Tech Circles; Unplugged Time; ScratchJr Time (Structured Challenges and Expressive Explorations); Word Time; and Closing Tech Circle (see [Appendix C](#) for details). The second-grade CAL-ScratchJr curriculum is centered around two books: a non-fiction book, *Grace Hopper: Queen of Computer Code*, which tells the story of a programmer from an underrepresented group in STEAM, and a fiction book, *Stellaluna*, with a social emotional focus and a clear sequence in the narrative. In addition to incorporating stories and literacy practices, the curriculum also includes developmentally appropriate practices for young children such as singing, dancing, and acting across the curriculum ([Singer, 2013](#)), in service of developing socioemotional skills and positive behaviors, as identified in the Positive Technological Development Framework ([Bers, 2012](#); [Bers et al., 2023](#)).

The curriculum is centered around seven *powerful ideas* of computer science, and seven *powerful ideas* of literacy, meaning a central concept or skill within a field that one can understand and utilize in the current domain, as well as apply to other domains ([Bers, 2017](#); [Bers et al., 2023](#)). The *powerful ideas* of CS and literacy are put in parallel to emphasize the connection between the two domains and to strengthen one another. For example, in Lesson 2, children begin with a word activity about sequencing, in which they rearrange the words *Cat, Is, On, The, Mat* and discuss how rearrangement changes the meaning of a sentence. The lesson then transitions into a computer science-focused discussion about how algorithms are the same: a sequence of symbols that change their meaning in different orders. Appendix D outlines the various *powerful ideas* in the curriculum.

[Table 3](#) presents the lesson dosage in each classroom. Due to the COVID pandemic, technology issues, and time constraints, not all classrooms completed the entire 24-lesson curriculum. Six out of the nine intervention classrooms completed all 24 lessons in the curriculum, which is 100% dosage. Two classrooms completed 19 out of 24 lessons (79% dosage), and one classroom implemented 11 out of 24 lessons (46% dosage). Nevertheless, most participating students who were assessed at the beginning were also assessed after the curriculum intervention in May and June 2022. [Appendix E](#) states the fidelity of implementation.

Table 3

Lesson dosage in each implementation classroom.

School ID	Implementation Teacher ID	Classroom ID	Lesson Dosage
1	1	1	24
1	1	2	24
2	2	3	19
2	2	4	19
3	3	5	11
5	4	6	24
5	5	7	24
5	6	8	24
6	7	9	24

2.5. Data analysis

All analyses conducted in this article use the multilevel model of the general form,

Level-1: Student Level

$$PostY_{ij} = \beta_{0j} + \beta_{1j} * PreY_{ij} + \sum_{m=1}^M \beta_{2mj} X_{mij} + \varepsilon_{ij}$$

Level-2: Cluster (School) Level

$$\beta_{0j} = \gamma_{00} + \gamma_{01} * Condition_j + \gamma_{02} * School_{pr eY_j} + \gamma_{03} * School_{SES} + \mu_{0j}$$

Where in this model: *PostY_{ij}* indicates the outcome variable; *Condition_j* indicates which schools were assigned to the treatment; *PreY_{ij}* is student-level pre-assessment of the outcome variable; *X_{mij}* indicates student-level covariates including gender (Female or Male), Individualized Educational Plan status (IEP, Yes or No), Limited English Proficiency status (LEP, Yes or No), and Free/reduced lunch status (Yes, or No). The inclusion of these demographic variables, which have been found to affect student outcomes, is theory-driven. Therefore, no competing models are examined. *School_{pr eY_j}* is the school-level pre-assessment of the outcome, and *School_{SES}* is the school-level socio-economic status, which is measured by the percentage of free/reduced lunch students. Variance of

ε_{ij} is the within school variation in student residuals, and variance of μ_{0j} is the between-school residual variation. The Restricted Maximum Likelihood is used due to the small number of clusters (Kenward & Roger, 1997). All data analysis was conducted using R version 4.2.2.

The school level was used instead of the classroom level because the cluster randomized control trial was done at the school level and because some classrooms were taught by the same teacher. Additionally, given the sample size in the current study and our investigation needs, we opted not to use a three-level model that includes a person level. A two-level model meets our investigation's objective of examining the treatment impact, whereas a three-level model, requiring a larger sample size and more clusters, would be more complicated to interpret. Furthermore, as our design is experimental, not including individual as another level for the repeated measure will not affect the results, as intra-individual variability is random across both treatment and control conditions.

In consideration of the assumptions for our analyses, histograms of Level-1 residuals for both the model predicting CSA and the model predicting *TechCheck* indicate a normal distribution. Furthermore, we confirmed that Level-1 residuals are independent from level-1 predictors in each model. Despite the small school-level sample, school-level intercept residuals remain independent from school-level predictors in predicting CSA. Notably, school-level covariates were not included when predicting *TechCheck* due to limited variability between schools (see Fig. 5).

3. Results

3.1. Descriptive statistics

Tables 4 and 5 present the descriptive statistics (sample size, mean, and standard deviation) of students' coding performance in ScratchJr as measured by CSA, and their computational thinking as measured by the unplugged assessment, *TechCheck*, in each condition. On average, both the treatment and the control group increased from the baseline coding performance. However, the treatment group showed a much larger increase in coding performance compared to the control group. Regarding computational thinking, no notable difference between the treatment and control groups was shown, while slightly higher scores after the intervention were shown by both groups.

3.2. Baseline equivalence

Table 6 presents student participants' demographic characteristics by the condition (treatment and control). The two conditions generally have comparable student demographics, except for the Limited English Proficiency (LEP) percentage, in which the treatment group consisted of a higher percentage of LEP students. To examine the baseline equivalence of students' coding proficiency and computational thinking, students' baseline CSA and *TechCheck* were used as the predicted variables, respectively, and the condition variable was used as a predictor while allowing school-level intercepts to vary to adjust for the clustering effect. Tables 7 and 8 present the descriptive statistics of the baseline CSA and *TechCheck* measures with the school-level clustering adjustment. The CSA baseline difference between treatment and control is 0.22 (Cohen's $d = 0.03$), and the *TechCheck* baseline difference between

Table 4
Descriptive statistics of coding performance Measure–CSA by Condition.

Outcome measure	Treatment			Control		
	n	Mean	SD	n	Mean	SD
Pre-CSA	149	4.62	3.39	164	4.50	2.71
Post-CSA	149	15.68	8.32	164	7.23	4.49

Note. The statistics are based upon student-level data.

Table 5
Descriptive statistics of computational thinking Measure–TechCheck by Condition.

Outcome measure	Treatment			Control		
	n	Mean	SD	n	Mean	SD
Pre-TechCheck	146	6.99	2.53	162	7.18	2.46
Post-TechCheck	146	8.60	2.91	162	8.90	2.63

Note. The statistics are based upon student-level data.

Table 6
Descriptive statistics of demographic characteristics by Condition.

Background Characteristics	% in Control	% in Treatment
Female	48.5	50.3
Limited English Proficiency (LEP)	5.3	17.2
Individualized Education Plan (IEP)	10.5	9.6
Free/reduced-price lunch	18.1	24.2

Table 7
Coding performance baseline descriptive statistics.

Treatment				Control			
# schools	# students	Mean	SE	# schools	# students	Mean	SE
5	149	4.88	0.60	6	164	4.66	0.41

Table 8
Computational thinking baseline descriptive statistics.

Treatment				Control			
# schools	# students	Mean	SE	# schools	# students	Mean	SE
5	146	7.08	0.48	6	162	7.12	0.32

treatment and control is -0.04 (Cohen's $d = -0.01$). These statistics suggest that baseline equivalence is met between the treatment and control groups for both the coding proficiency in ScratchJr as measured by CSA, and the computational thinking, as measured by *TechCheck*.

3.3. Impact results

3.3.1. RQ1: what is the impact of the CAL-ScratchJr curriculum in second-grade classrooms on the development of students' coding skills?

To address our first research question, we used multilevel modeling to examine how students' post-CSA scores vary by treatment and control condition. In the unconditional model (without including predictors for any fixed effects), preliminary analysis of within- and between-groups variance suggested that the between-school variance was large ($\rho = 0.31$), which indicates that 31% of the variation in the post-CSA scores was due to their school membership.

Additionally, Fig. 4 displays the random effect dot plot of school post-CSA intercepts. The plot shows the deviation from the grand mean of student post-CSA for each school, with dots representing each school's estimated intercept and error bars indicating corresponding standard error. Y-axis numbers denote school ID (also see Table 3). Variability in intercept between schools suggests a need for school cluster adjustment and controlling for school-level covariates.

The impact model for post-CSA included the random intercept effect and fixed effect from the treatment variable while controlling for pre-CSA, demographic variables, school-level pre-CSA, and free-reduced lunch percentage as covariates. The unstandardized coefficient (B) for treatment in Table 9 is 8.05, indicating a significant difference in post-CSA scores between treatment and control groups after controlling for

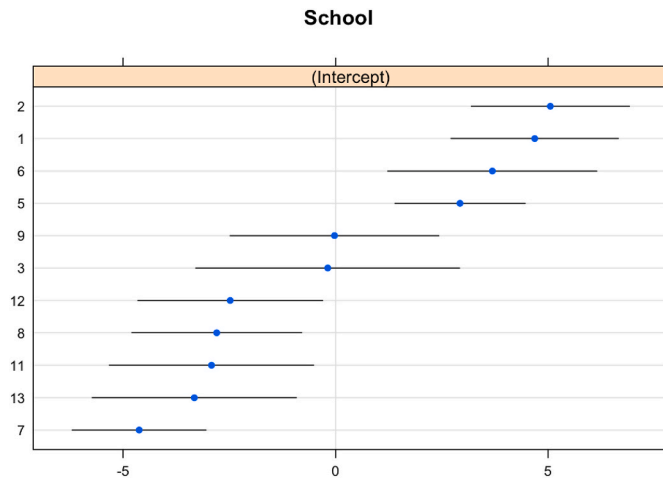


Fig. 4. The dot plot of random effects: School intercept of post-coding performance.

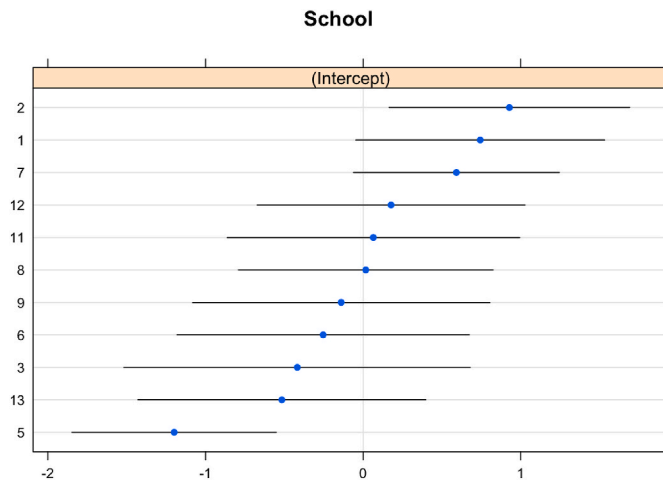


Fig. 5. The dot plot of random effects: School intercept of post-computational thinking.

Table 9

Fixed effects of treatment on post-coding performance (Post-CSA) controlling for covariates.

Fixed effects	<i>B</i>	<i>SE B</i>	<i>p</i> value
Intercept	8.13	4.35	0.11
Condition (Treatment)	8.05	1.10	<0.001
Pre-CSA	0.87	0.11	<0.001
Gender (Female)	0.67	0.69	0.33
IEP (Yes)	-2.34	1.17	0.05
LEP (Yes)	-0.96	1.32	0.47
Free/Reduced Lunch (Yes)	0.40	0.96	0.68
School level Pre-CSA	-1.30	1.17	0.31
School % of Free/Reduced Lunch	-2.0	3.16	0.56

covariates. This difference has a medium effect size ($p < 0.001$, Cohen's $d = 0.68$) and suggests that the CAL-ScratchJr curriculum had a positive impact on second-grade students' coding skills.

3.3.2. RQ2: what is the impact of the CAL-ScratchJr curriculum in second-grade classrooms on the development of students' computational thinking?

To address our second research question, we used multilevel modeling to examine how students' post-TechCheck scores vary by treatment and control condition. In the unconditional model (without

controlling for any fixed effect), preliminary analysis of within- and between-groups variance suggested that the between-school variance was small ($ICC = 0.07$), which indicates that 7% of the variation in the post-TechCheck scores was due to students' school membership.

Additionally, Fig. 5 displays the random intercept effect of school post-TechCheck, with dots representing each school's estimated intercept and error bars indicating the standard error. There is limited variability between schools. The post-TechCheck impact model included the random intercept effect and fixed effect from the condition variable while controlling for student-level pre-CSA and demographic variables. School-level covariates were not included due to limited variability between schools. Appendix F contains the correlations between CSA and TechCheck at the pre- and post-time points, as well as the correlations of CSA and TechCheck across two time points for both treatment and control conditions.

Table 10 presents the fixed effects of treatment on students' post-TechCheck after controlling for the baseline and student-level demographic covariates. The results indicate that students' computational thinking, as measured by post-TechCheck, did not differ significantly between the treatment group and the control group after the CAL-ScratchJr curriculum implementation with an unstandardized coefficient of -0.1 (labeled as *B* in Table 10, $p = 0.81$, Cohen's $d = -0.02$). The results imply that implementation of the CAL-ScratchJr curriculum did not make a significant difference regarding students' computational thinking.

4. Discussion

4.1. Impact of the CAL-ScratchJr curriculum on coding performance

The results indicate that, compared to the control group who did business as usual, the intervention showed a medium positive impact on students' coding performance (Cohen's $d = 0.68$). This result suggests that the CAL-ScratchJr curriculum implementation among second graders is successful in introducing students to basic computer science concepts (e.g., algorithm, control structure, and modularity). The result provides empirical experimental evidence that implementing CS and programming curricula among young students can be successful. Because multiple school districts participated in the study, the results of the study can be generalized to similar schools in the United States.

The medium positive impact of the CAL-ScratchJr curriculum on students' coding proficiency indicates that its pedagogy and content have effectively enhanced students' coding skills. For example, as shown in Table 1, CAL-ScratchJr teaches students coding through open-ended projects, fostering an open mindset by allowing them to create their own projects rather than following a predetermined path. This method helps avoid discouragement that may arise from not reaching specific milestones, and instead promotes autonomy and motivation through personally meaningful projects (Bers et al., 2023; Yang & Bers, 2023). Furthermore, the CAL-ScratchJr curriculum incorporates seven powerful ideas in computer science, drawing parallels between these ideas and natural language writing (see Appendix D). This connection aids students in understanding the relationship between coding and writing, making it more relatable. The positive impact suggests that the

Table 10

Fixed effects of treatment on post-computational thinking (Post-TechCheck) controlling for covariates.

Fixed effects	<i>B</i>	<i>SE B</i>	<i>p</i> value
Intercept	4.96	0.54	<0.001
Treatment	-0.10	0.38	0.81
Pre-TechCheck	0.55	0.06	<0.001
Gender (Female)	0.26	0.27	0.34
IEP (Yes)	-0.99	0.46	0.03
LEP (Yes)	-0.15	0.48	0.75
Free/Reduced Lunch (Yes)	-0.07	0.35	0.84

CAL-ScratchJr curriculum's pedagogy and content are viable, and can serve as a reference for designing similar CS curricula.

4.2. Impact of the CAL-ScratchJr curriculum on computational thinking

The study also investigated the impact of the CAL-ScratchJr curriculum on second graders' computational thinking. No notable difference was found between the treatment and control groups. As section 1.2 notes, the literature shows mixed results on this topic. Comparing the result with previous studies helps explain the variability in outcomes. This study adds empirical evidence from a cluster RCT involving multiple schools and sites, offering possible explanations for the mixed results.

When using the same operational definition of computational thinking (*TechCheck*), the studies by [Relkin et al. \(2021, 2022\)](#), [Yang et al. \(2023\)](#) and the current study consistently reported no significant effect on students' computational thinking from the CAL-ScratchJr, codeSpark Academy, and CAL-KIBO robotics curricula. Although [Relkin et al. \(2021\)](#) found a positive effect for the CAL-KIBO robotics curriculum among first graders, this was confounded by a higher baseline computational thinking score and is not comparable to the second grade results in the current study.

Additionally, [Grillo-Hill et al. \(2019\)](#), [Straw et al. \(2017\)](#) and the current study, using different operational definitions of computational thinking and different curricula, consistently reported no significant effect on students' computational thinking.

In contrast, [Oluk and Çakir \(2021\)](#) reported a significant positive effect on sixth graders' computational thinking using the Computational Thinking Skill Levels Scale by [Korkmaz et al. \(2015\)](#) and the *Code.org* curriculum. However, their study lacked detailed descriptions of the quasi-experiment, raising concerns about replicability. The unplugged activities in the *Code.org* curriculum, which closely resemble many computational thinking instruments, could also potentially enhance scores due to the testing effect, alongside the inherent differences between these studies.

In summary, the non-significant effect on students' computational thinking in this study aligns with findings from other studies with various CS curricular intervention, both using the same or different instruments to measure computational thinking. This consistency suggests a null effect. Conversely, the positive effects reported for *Code.org* may be due to a testing effect from similarities between its unplugged activities and the computational thinking instruments.

Several factors could explain the non-significant impact on computational thinking. First, knowledge gained from concrete programming experiences and real problem-solving may differ from abstract computational thinking concepts. According to [Piaget's \(1971\)](#) theory, second-grade students are in the concrete operational stage and may not yet be ready for abstractions. Children can create coding projects but may not recognize the generalized patterns behind them. [Guzdial \(2015\)](#) noted that students need to see many concrete examples before abstracting underlying principles. Therefore, the non-significant difference in computational thinking scores between the treatment and control could be due to emerging abstraction skills.

Second, the study may not have enough statistical power to detect the impact on computational thinking due to the small number of schools. Typically, around 30 schools are needed to detect a medium effect size in cluster RCT with two levels. Including more schools may be necessary to verify the impact on computational thinking.

Third, the curriculum implementation time may not be long enough to determine its effect on students' computational thinking. Due to technology issues, time constraints, and disruptions from the COVID-19 pandemic, not all teachers implemented all 24 lessons from the curriculum. Further research is necessary to examine how various lengths of the intervention may impact on computational thinking.

Last but not the least, the *TechCheck* assessment, comprising isolated problems on computational thinking (e.g., decomposition), may

represent a distinct concept compared to real-world coding scenarios where students apply computational thinking. In the context of isolated assessments, a minor error could lead to an incorrect result, potentially overshadowing students' understanding of the underlying concept. However, in practical problem-solving situations, students demonstrate their application of computational thinking. For instance, when identifying the beginning, middle, and end scenes of *Stellaluna* in the CAL-ScratchJr curriculum, students engage in meaningful decomposition. While students might proficiently identify components within a narrative, they may struggle with visually decomposing abstract geometric shapes. This nuanced distinction emphasizes the importance of considering the practical application of computational thinking skills beyond the confines of isolated assessments.

The results in this study indicated that young students' coding skills can improve significantly without significant changes in their computational thinking. This suggests that while computational thinking is valuable, it should be developed gradually through hands-on coding projects. The focus should be on practical coding first, with the resulting skills following from long-term engagement, rather than the other way around. Educators and curriculum designers should avoid abstract, isolated computational thinking activities that lack real-world application for early elementary students, as these practices may not be developmentally appropriate.

[Resnick and Rusk \(2020\)](#) also cautioned researchers not to downplay the value of coding, nor focus too narrowly on teaching computational thinking out of context. They advocate for creating projects and expressing ideas through coding, as this provides meaningful context and motivation. Rather than focusing solely on improving computational thinking, future CS curriculum developers for younger students might consider how to engage students in coding as a means to express themselves and create projects in personally meaningful contexts.

4.3. Limitations and future research

Several limitations in the study need to be considered in the interpretation of the findings. First, implementation levels varied among classrooms due to the challenges of finding sufficient time, leading to differences in the amount and number of CAL-ScratchJr lessons implemented and integrated into teacher schedules. Furthermore, we could not control for students' extracurricular or home activities that may have had bearing on the intervention effect. Another limitation is that the study assessed the effect of the intervention only during the intervention term. Future research could follow up with the students to examine the preservation, acceleration, or decline of the intervention effect over time. Additionally, longitudinal research could compare students' computational thinking over a longer exposure period to the CAL-ScratchJr curriculum to that of students not exposed to the intervention.

5. Conclusion

The CAL-ScratchJr curriculum implementation showed a medium positive effect on the second-grade participants' coding skills, which suggests that the CAL-ScratchJr curriculum implementation is successful in improving students' coding skills. However, no notable difference was found in terms of computational thinking compared to that of their counterparts in the control group. The distinct effects on coding skills and computational thinking may suggest that coding and computational thinking are not causally related among second-grade students. Transferring programming knowledge from concrete experience and problem-solving to abstract computational thinking may take some time for young learners, as they are still in the concrete operational stage. The study presented in this paper is one of the few in the extant literature that conducted a cluster RCT to examine the impact of a CS curriculum intervention among second graders in a real classroom setting in the U.S. The results in this study remind educators, curriculum designers and

researchers that further studies are needed to elucidate the relationship between coding and computational thinking, particularly at early ages.

CRediT authorship contribution statement

Zhanxia Yang: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Jessica Blake-West:** Writing – review & editing, Writing – original draft, Validation. **Dandan**

Yang: Writing – review & editing, Writing – original draft. **Marina Bers:** Writing – review & editing, Validation, Supervision, Funding acquisition, Conceptualization.

Acknowledgements

Funding: This work was supported by the United States Department of Education [grant # U411C220202, 2022].

Appendix A

There are a total of 16 students who attrited for both CSA and TechCheck measures. Among these 16 students, seven are female, five are LEP, two have IEP, and five receive free/reduced-price lunch. The table below presents a comparison of the characteristics of these attrited students between the two tested groups.


Appendix A

Table of a comparison of the characteristics of attrited students between the two conditions.

	Treatment (n = 11)	Control (n = 5)
Female	5	2
LEP	5	0
IEP	0	2
Free/reduced-price lunch	4	1

Appendix B

Although both CSA, measuring ScratchJr coding skills, and *TechCheck*, measuring computational thinking, are available online upon request from DevTech Research Group at Boston College, we included sample questions that were cited already in this study as well as three others we received permission from the research group.



Q3.1: Make this program.
Please change the program so that Cat will make a pop noise 4 times, and move forward 4 times, but **only** say Hi **once**.
Rephrase: Right now Cat says hi 4 times, how can I change it to only say Hi one time?

☐ Satisfactory: Child moves hi block outside of repeat loop

☐ Unsatisfactory: Child does not move the hi block out of the repeat loop or removes the entire repeat loop without changing parameters appropriately

Fig. B1. An example question of CSA in Fluency stage of CSA.
Note. This figure is from “Examining gender difference in the use of ScratchJr in a programming curriculum for first graders.” By Z. Yang & Bers, 2023, *Computer Science Education*, p.6. Copyright 2024 by Informa UK Limited.

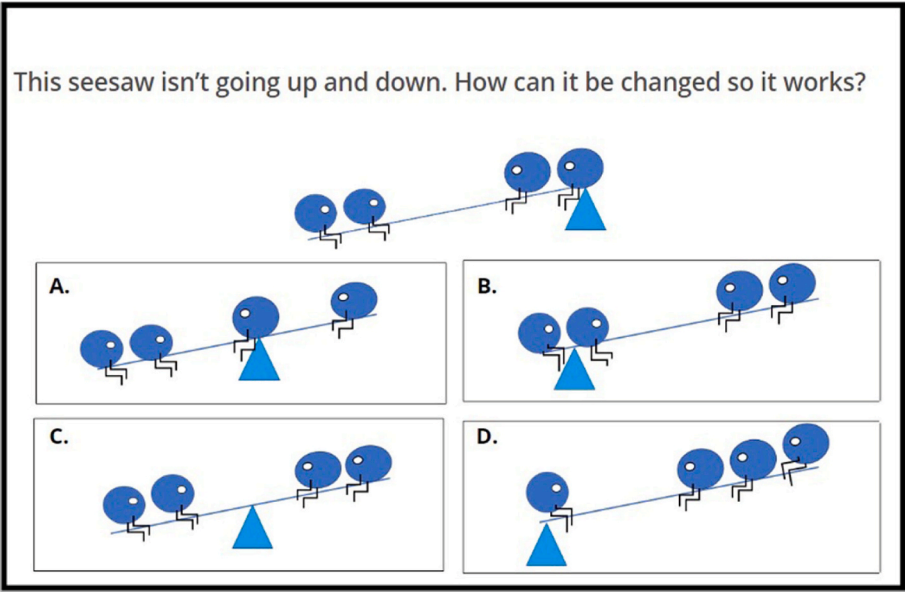


Fig. B2. TechCheck symmetry problem question designed to probe debugging skills.
Note. This figure is from “TechCheck: Development and validation of an unplugged assessment of computational thinking in early childhood education.” By E. Relkin, L. de Ruiter, and M. U. Bers, 2020, *Journal of Science Education and Technology*, 29(4), p.486. Copyright 2024 by Springer Nature B.V.

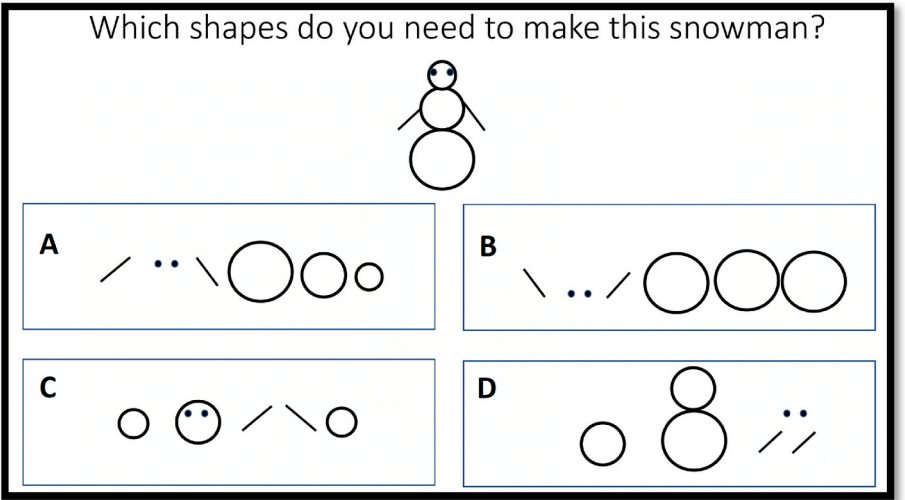


Fig. B3. TechCheck object decomposition question, an example of modularity skills (DevTech Research Group).

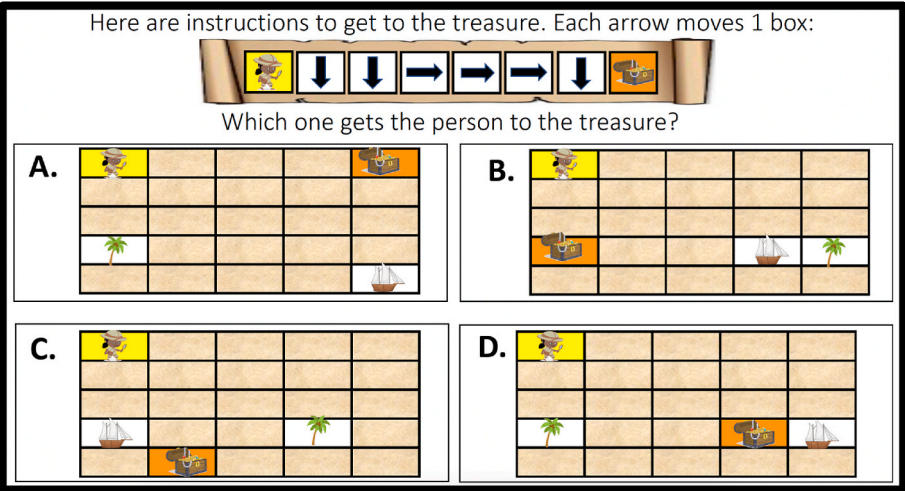


Fig. B4. TechCheck map abstraction question, an example of control structure skills (DevTech Research Group).

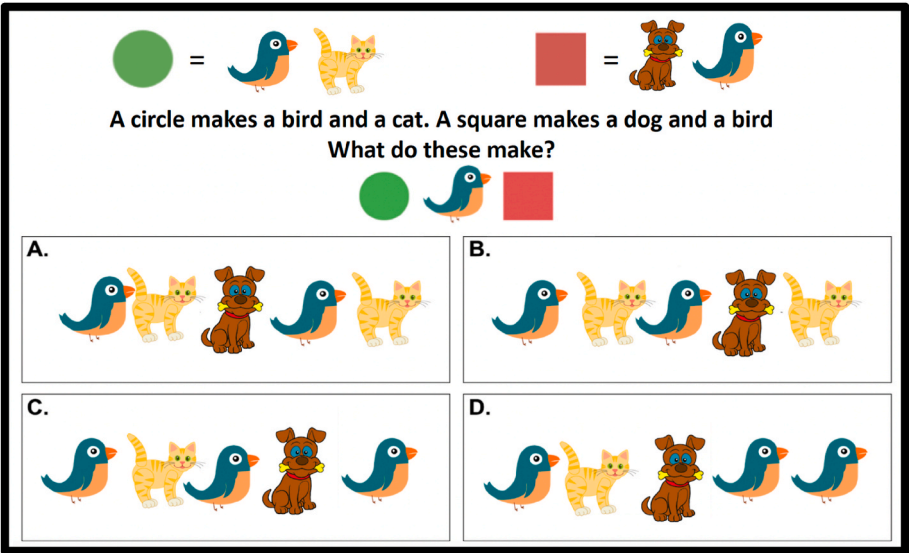


Fig. B5. TechCheck symbol shape puzzles question, an example of representation skills (DevTech Research Group).

Appendix C

An Example CAL-ScratchJr Lesson Format.

Grade 2, Lesson 2 of 24

Coding as Another Language - ScratchJr
(CAL-ScratchJr)

DEVTECH RESEARCH GROUP

Lesson 2: Order Matters

Lesson 2 Overview: Lesson 2 introduces the concept of representation and that symbols can stand for something else. Children explore introductory algorithms and sequencing using cut-out ScratchJr blocks.

Powerful Ideas from Computer Science: Algorithms, Representation
Powerful Ideas from Literacy: Sequencing, Alphabet and Letter-Sound Correspondence

I. Warm Up: Did that Sentence Make Sense?

- Explain the need for order in languages. Use cut out words of the sentence ("The cat is on the mat"), scramble them, and read out the scrambled sentence (e.g., "mat the on is cat the"). Ask children if it makes sense and why/why not.

II. Opening Tech Circle: What is an Algorithm?

- Explain that the instructions we give computers are called "algorithms." An algorithm is a sequence of steps in the right order.
- Collect examples of activities that need to be done in a certain order (e.g., brushing your teeth, putting on socks and shoes).

III. Unplugged Time: Program the Teacher

- Tell children that the teacher will now be the computer and they will get to program them! Remind them that they need to say all the steps in the right order. Children will be responsible for verbally directing their teacher to special destinations in the classroom (e.g., to a bookcase or a closet) or doing a task (e.g., making a sandwich).
- Discuss how important it is to be specific and how important order is in programming.

IV. ScratchJr Time: ScratchJr Blocks

- Using ScratchJr Block Cut Outs, ask children "how do you know what ScratchJr blocks mean?" Focus on the representation of color (e.g., blue = motion). Discuss the importance of the Green Flag and the End blocks.

V. Closing Tech Circle: Roses & Buds

- Each child shares one "rose" (thing that they learned) and one "bud" (thing they want to know more about and are excited about learning).

Vocabulary

Algorithm
Program
Programmer

ScratchJr Blocks:

Start on Green Flag
Motion Blocks
Go to Start
End

12

Appendix D

Appendix D

Powerful ideas included in the second grade CAL-ScratchJr curriculum.

Powerful Ideas from Computer Science	Powerful Ideas from Literacy	Connecting the Powerful Ideas
Algorithms Relevant Lessons: 2, 4, 5, 6, 11, 12, 13, 14, 16, 17, 19, 20, 22, 23, 24	Sequencing Relevant Lessons: 2, 4, 5, 6, 10, 11, 12, 13, 17, 20, 24	Emphasis on “order matters,” and that complex tasks can be broken down into step-by-step instructions in a logical way.
Design Process Relevant Lessons: 3, 5, 6, 7, 8, 11, 12, 13, 16, 19, 20, 21, 22, 23, 24	Writing Process Relevant Lessons: 3, 5, 8, 10, 11, 21, 22, 23, 24	Creative, iterative, cyclic processes that involve imagining, planning, making, revising, and sharing, with different starting points.
Representation Relevant Lessons: 1, 2, 3, 17, 19	Alphabet and Letter-Sound Correspondence Relevant Lessons: 2	Symbols have different attributes (color, shape, sound, etc.) in order to represent something else.
Debugging Relevant Lessons: 7, 8, 11, 22	Editing and Audience Awareness Relevant Lessons: 4, 6, 7, 8	Systematic analysis, testing, and evaluation to improve communication to the intended audience (computer or person). Whenever miscommunication occurs, the programmer or writer uses a variety of strategies to solve the problem.
Control Structures Relevant Lessons: 9, 12, 13, 14, 15, 16, 18	Literary Devices Relevant Lessons: 9, 12, 13, 14, 15, 16, 18, 19	Advanced strategies to communicate a set of ideas using repetition, patterns, conditionals and events.
Modularity Relevant Lessons: 4, 10, 16, 20, 21	Phonological Awareness Relevant Lessons: 5, 9, 21	Decomposition, or breaking down a complex task into smaller tasks and re-using those new modules.
Hardware/Software Relevant Lessons: 3, 10	Tools of Communication and Language Relevant Lessons: 1, 3, 10, 12, 14, 17	Communicating abstract ideas through tangible means. Just like hardware and software work together, the expression of thoughts through language requires a medium for communicating to the outside world, such as spoken or written word.

Appendix E

Fidelity of Implementation

Implementation fidelity was ensured through various measures. All teachers underwent two 2-h training sessions with an experienced trainer. Additionally, each intervention school appointed a tech leader, a staff member familiar with the curriculum from a semester-long graduate course. These tech leaders supported teachers on-site, ensuring adherence to the curriculum and logistical aspects of the research, such as maintaining accurate lesson logs. Furthermore, designated members of the research team acted as virtual coaches. A dedicated school staff member served as the research coordinator, overseeing all research logistics. On top of all of these, a third party evaluator company, independent from the research team, monitors and evaluates all components of the research as part of the funding requirement.

Appendix F

Appendix F1

Table of correlations between two variables and time points in the treatment condition.

	Treatment			
Correlation Between Variables	Pre-CSA, Pre-TechCheck	Post-CSA, Post-TechCheck	Pre-CSA, Post-CSA	Pre-TechCheck, Post-TechCheck
Correlation Value	0.22**	0.38***	0.32***	0.53***

Appendix F2

Table of correlations between two variables and time points in the control condition.

	Control			
Correlation Between Variables	Pre-CSA, Pre-TechCheck	Post-CSA, Post-TechCheck	Pre-CSA, Post-CSA	Pre-TechCheck, Post-TechCheck
Correlation Value	0.17*	0.32***	0.59***	0.54***

References

- Arfe, B., Vardanega, T., & Ronconi, L. (2020). The effects of coding on children's planning and inhibition skills. *Computers & Education*, 148, 1–16.
- Ausubel, D. P. (1963). *The psychology of meaningful verbal learning*. New York: Gruene and Stratton.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54.
- Bers, M. U. (2008a). *Blocks to robots learning with technology in the early childhood classroom*. New York, NY: Teachers College Press.
- Bers, M. U. (2008b). Engineers and storytellers: Using robotic manipulatives to develop technological fluency in early childhood. In O. Saracho, & B. Spodek (Eds.), *Contemporary perspectives on science and technology in early childhood education* (pp. 105–125). Charlotte, NC: Information Age Publishing.
- Bers, M. U. (2012). *Designing digital experiences for positive youth development: From playpen to playground*. New York, NY: Oxford University Press.

- Bers, M. U. (2017). The Seymour test: Powerful ideas in early childhood education. *International Journal of Child-Computer Interaction*, 14, 10–14.
- Bers, M. U. (2019). Coding as Another Language: A pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education*, 6(4), 499–528.
- Bers, M. U. (2020). *Coding as a playground: Programming and computational thinking in the early childhood classroom* (2nd ed.). New York, NY: Routledge Press.
- Bers, M. U. (2021a). Coding, robotics and socio-emotional learning: Developing a palette of virtues PIXEL-BIT. *Revista de Medios y Educación*, 62, 309–322.
- Bers, M. U. (Ed.). (2021b). Teaching computational thinking and coding to young children. *IGI Global*.
- Bers, M. U. (2022). *Beyond coding: How children learn human values through programming*. Cambridge, MA: The MIT Press.
- Bers, M., & Cassell, J. (1999). Interactive storytelling systems for children: Using technology to explore language and identity. *Journal of Interactive Learning Research*, 9(2), 603–609.
- Bers, M. U., Blake-West, J., Kapoor, M. G., Levinson, T., Relkin, E., Unahalekhaka, A., & Yang, Z. (2023). Coding as another language: Research-based curriculum for early childhood computer science. *Early Childhood Research Quarterly*, 64, 394–404.
- Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, 72, 145–157.
- Bers, M. U., Levinson, T., Yang, Z., Rosenberg-Kima, R. B., Ben-Ari, A., Jacob, S., Dubash, P., Warschauer, M., Gimenez, C., Gonzalez, P., & Gonzalez, H. (2023). Coding as another language: An international comparative study of learning computer science and computational thinking in kindergarten. In P. Blikstein, J. van Aalst, & K. Brennan (Eds.), *17th International Conference of the Learning Sciences (ICLS)* (pp. 1659–1665). <https://2023.isls.org/proceedings/>.
- Blake-West, J. C., & Bers, M. U. (2023). ScratchJr design in practice: Low floor, high ceiling. *International Journal of Child-Computer Interaction*, 37. <https://doi.org/10.1016/j.ijcci.2023.100601>
- Brooks, B., & Phillips, R. (2017). The hour of code: Impact on attitudes towards self-efficacy with computer science. *Code.org*. Retrieved from <https://code.org/files/HourOfCodeImpactStudyJan2017.pdf>.
- Bruckman, A., Jensen, C., & DeBonte, A. (2002). Gender and programming achievement in a CSDL. In *Computer support for collaborative learning: Foundations for a CSDL community: Proceedings of CSDL 2002 boulder, Colorado, USA January 7–11, 2002* (pp. 119–127).
- Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *The National Association for Media Literacy Education's Journal of Media Literacy Education*, 4(2), 121–135.
- Burke, Q., & Kafai, Y. B. (2014). *Connected code: Why children need to learn programming*. Cambridge, MA: MIT Press.
- Century, J., Ferris, K. A., & Zuo, H. (2020). Finding time for computer science in the elementary school day: A quasi-experimental study of a transdisciplinary problem-based learning approach. *International Journal of STEM Education*, 7(1), 1–16.
- Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers in Education*, 109, 162–175.
- Code.org, CSTA, & ECEP Alliance. (2022). 2022 State of computer science education: Accelerating action through advocacy. <https://advocacy.code.org/stateofcs>.
- Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in <http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf>.
- Denniston, C., Roome, B. R., & Wilson, N. (2015). *Efficacy study of the foos game*. KnowProgress [White Paper].
- DePryck, K. (2016). From computational thinking to coding and back. In *Proceedings of the fourth international conference on technological ecosystems for enhancing multiculturalism* (pp. 27–29).
- DevTech Research Group. (n.d.). TechCheck: Computational thinking assessments. Boston College. <https://sites.bc.edu/devtech/assessments/techcheck/>.
- Falloon, G. (2016). An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. *Journal of Computer Assisted Learning*, 32(6), 576–593.
- Fayer, S., Lacey, A., & Watson, A. (2017). STEM occupations: Past, present, and future. *Spotlight on Statistics*, 1, 1–35.
- Flórez, F. B., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research*, 87(4), 834–860.
- Grillo-Hill, A., Mahoney, C., Chow, E., & Li, L. (2019). *StoryCoder classroom feasibility study*. WestEd [White Paper] https://edcuration.com/resource/vendor/348/codeSpark_Feasibility/20Memo_Draft.pdf.
- Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43.
- Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool Publishers.
- Kalelioglu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52, 200–210.
- Kapoor, M., Yang, Z., & Bers, M. (2022). Supporting early elementary teachers' coding knowledge and self-efficacy through virtual professional development. *Journal of Technology and Teacher Education*, 30(4), 461–491.
- Kenward, M. G., & Roger, J. H. (1997). Small sample inference for fixed effects from restricted maximum likelihood. *Biometrics*, 983–997.
- Kirschner, P. A., Stribos, J. W., Kreijns, K., & Beers, P. J. (2004). The social dynamics of collaborative knowledge building. *Journal of Computer Assisted Learning*, 20(5), 367–381.
- Korkmaz, Ö., Çakır, R., & Özden, M. Y. (2015). Bilgisayarca düşünme beceri düzeyleri ölçeğinin (BDBD) ortaokul düzeyine uyarlanması. *Gazi Eğitim Bilimleri Dergisi*, 1(2), 143–162.
- Lahullier, S. C. (2019). *Code. Org and computational thinking: A mixed-Methods study on fifth-grade elementary school students*. New Jersey City: University.
- Laurent, M., Crisci, R., Bressoux, P., Chaachoua, H., Nurra, C., de Vries, E., & Tchounikine, P. (2022). Impact of programming on primary mathematics learning. *Learning and Instruction*, 82, 1–9, 101667.
- Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51–61.
- Manches, A., & Plowman, L. (2017). Computing education in children's early years: A call for debate. *British Journal of Educational Technology*, 48(1), 191–201.
- Mayer, R. E. (2002). Rote versus meaningful learning. *Theory and Practice*, 41(4), 226–232.
- Oluk, A., & Çakır, R. (2021). The effect of code.org activities on computational thinking and algorithm development skills. *Journal of Teacher Education and Lifelong Learning*, 3(2), 32–40.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
- Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, 1(1), 95–123.
- Piaget, J. (1971). The theory of stages in cognitive development. In D. R. Green, M. P. Ford, & G. B. Flamer (Eds.), *Measurement and Piaget*. McGraw-Hill.
- Relkin, E. V. (2022). *The development of computational thinking skills in young children*. Tufts University. Doctoral dissertation.
- Relkin, E., & Bers, M. (2021). TechCheck-K: A measure of computational thinking for kindergarten children. In *2021 IEEE Global engineering education conference (EDUCON)* (pp. 1696–1702). IEEE.
- Relkin, E., de Ruiter, L., & Bers, M. U. (2020). TechCheck: Development and validation of an unplugged assessment of computational thinking in early childhood education. *Journal of Science Education and Technology*, 29(4), 482–498.
- Relkin, E., de Ruiter, L. E., & Bers, M. U. (2021). Learning to code and the acquisition of computational thinking by young children. *Computers & Education*, 169, Article 104222.
- Relkin, E., Johnson, S. K., & Bers, M. U. (2023). A normative analysis of the TechCheck computational thinking assessment. *Educational Technology & Society*, 26(2), 118–130.
- Resnick, M., & Rusk, N. (2020). Coding at a crossroads. *Communications of the ACM*, 63(11), 120–127.
- Saez-Lopez, J. M., Roman-Gonzalez, M., & Vazquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, 97, 129–141.
- Singer, E. (2013). Play and playfulness, basic features of early childhood education. *European Early Childhood Education Research Journal*, 21(2), 172–184.
- Straw, S., Bamford, S., & Styles, B. (2017). *Randomised controlled trial and process evaluation of code clubs*. National Foundation for Educational Research and the Raspberry Pi Foundation.
- Sullivan, A., & Bers, M. U. (2013). Gender differences in kindergarteners' robotics and programming achievement. *International Journal of Technology and Design Education*, 23, 691–702.
- Unahalekhaka, A., & Bers, M. U. (2022a). *Evaluating young children's creative coding: Rubric development and testing for ScratchJr projects*. Educ Inf Technol.
- Unahalekhaka, A., & Bers, M. U. (2022b). Clustering young Children's coding project scores with Machine learning. In *2022 IEEE global engineering education conference (EDUCON)* (pp. 79–85).
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.
- Wing, J. M. (2011). *Research notebook: Computational thinking—What and why? The link magazine*. Pittsburgh: Spring. Carnegie Mellon University. <https://www.cs.cmu.edu/link/research-notebookcomputational-thinking-what-and-why>.
- Woo, K., & Falloon, G. (2022). Problem solved, but how? An exploratory study into students' problem solving processes in creative coding tasks. *Thinking Skills and Creativity*, 46, Article 101193.
- Yadav, A., Good, J., Voogt, J., & Fisser, P. (2017). Computational thinking as an emerging competence domain. In M. Mulder (Ed.), *Competence-based vocational and professional education: Bridging the worlds of work and education* (pp. 1051–1067).
- Yang, D., & Bers, M. U. (2024). *Coding as Another Language: Impact on Math and Literacy Achievement in Early CS*. Philadelphia, PA: American Educational Research Association (AERA) Annual Meeting.
- Yang, D., Yang, Z., & Bers, M. U. (2023). The efficacy of a computer science curriculum for early childhood: evidence from a randomized controlled trial in K-2 classrooms. *Computer Science Education*, 1–21.
- Yang, Z., & Bers, M. (2023). Examining gender difference in the use of ScratchJr in a programming curriculum for first graders. *Computer Science Education*, 1–22.
- Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, 141, Article 103607.