Early Childhood Research Quarterly 64 (2023) 394–404

Contents lists available at ScienceDirect

Early Childhood Research Quarterly

journal homepage: www.elsevier.com/locate/ecresq

Coding as another language: Research-based curriculum for early childhood computer science ☆

Marina Umaschi Bers[a,*], Jessica Blake-West[a], Madhu Govind Kapoor[b], Tess Levinson[a], Emily Relkin[c], Apittha Unahalekhaka[b], Zhanxia Yang[a]

[a] DevTech Research Group, Lynch School of Education and Human Development, Boston College, 140 Commonwealth Ave, Chestnut Hill, MA 02467, USA
[b] Eliot-Pearson Dept of Child Study and Human Development, Tufts University, 105 College Ave, Medford, MA 02155, USA
[c] Educational Development Center, 300 5th Ave Suite 2010, Waltham, MA 02451, USA

ARTICLE INFO

Keywords:
Design-based research
Coding
Computational Thinking
Powerful Ideas
ScratchJr

ABSTRACT

This paper describes the iterative research and evaluation of the Coding as Another Language (CAL) curriculum that utilizes the free ScratchJr programming language in kindergarten to second grade. CAL was designed using principles of three theoretical frameworks: Curriculum Research Framework (CRF), which proposes different phases in the creation of research-based curriculum; Constructionism, which presents a computationally-rich project-based methodology based on identifying powerful ideas from a learning domain; and Positive Technological Development, which intentionally integrates socio-emotional and ethical dimensions into curricular experiences. The pedagogical foundation of CAL involves the understanding of coding as a literacy, that is, putting developmentally-appropriate powerful ideas of computer science in conversation with those taught in language arts. The paper first describes CAL and then presents results from both a pilot study and a cluster randomized controlled trial that set to evaluate CAL's feasibility and impact on students' learning outcomes. Our findings showed that the CAL curriculum was not only feasible to implement, but also effective for improving coding skills. However, CAL's impact on computational thinking is less clear given that in the cluster randomized controlled trial, both the control and the intervention groups improved equally on a measure of computational thinking.

1. Introduction

Educators, researchers, and policymakers are recognizing the need to introduce computer science (CS) to children starting at an early age. In recent years, the focus has expanded from programming skills (i.e., learning to code) to also include cognitive abilities known as computational thinking (CT), a broad set of universally applicable analytic and problem-solving skills, dispositions, and habits rooted in computer science. Examples of CT skills and abilities include thinking recursively, using abstraction, and applying heuristic reasoning (Aho, 2012; Bers et al., 2021; Barr & Stephenson, 2011; Lodi & Martini, 2021; Grover & Pea, 2013; Wing, 2006, 2011). In addition, developmentally appropriate programming languages for young children, previously introduced as prototypes, have expanded to serve as functional and widely accessible tools, evaluated for use in multiple contexts (Bers & Resnick, 2015; Sullivan & Bers, 2019).

This paper describes the Coding as Another Language (CAL) curriculum which uses the free introductory ScratchJr programming language and integrates the teaching of early literacy with early computer science. This work addresses the research question whether and how an early childhood computer science curriculum can be developed following both a theoretical and evidence-based approach. In order to address the question, the paper introduces three theoretical frameworks upon which CAL was iteratively designed and evaluated: the Curriculum Research Framework (CRF) developed by Clements (2007), Constructionism proposed by Papert (1980), and Positive Technological Development (PTD) created by Bers (2012). Second, it describes the core innovation of the CAL curriculum, the integration of early literacy and computer science by identifying overlapping developmentally appropriate core concepts and skills – the powerful ideas covered in the curriculum. Finally, it presents two sets of results: from a pilot study, which explored CAL's feasibility and initial student learning outcomes regarding coding

☆ The findings reported in this manuscript have not been previously published, and the manuscript is not being simultaneously submitted elsewhere. My coauthors and I do not have any interests that might be interpreted as influencing the research, and APA ethical standards were followed in the conduct of the study.

* Corresponding author.
E-mail address: Marina.bers@bc.edu (M.U. Bers).

https://doi.org/10.1016/j.ecresq.2023.05.002
Received 15 December 2021; Received in revised form 12 April 2023; Accepted 9 May 2023

and computational thinking, and from a cluster randomized controlled trial which interrogated the relationship between computational thinking and coding.

2. Three theoretical frameworks

Three frameworks grounded the development of CAL. First, the CRF for creating evidence-based curriculum (Clements, 2007) informed the different phases of the research conducted over a decade and a half resulting in CAL. Second, Papert's Constructionism (Papert, 1980) positions CAL as a computationally-rich design-based curriculum focused on hands-on, project-based learning for young children to program meaningful projects with ScratchJr. Third, the PTD framework developed by Bers (2012) provides a lens to incorporate socio-emotional and ethical dimensions alongside computer science knowledge and skills in every unit. Each of these three frameworks will be described as they informed CAL.

2.1. Curriculum research framework

Work in early childhood has emphasized the importance of evidence-based curricular materials (Clements, 2007; Code.org, CSTA, & ECEP Alliance, 2021; Manches & Plowman, 2017; Tucker et al., 2003). While computer science curricular units have been developed with varying levels of adaptation and success (Century, Ferris & Zuo, 2020; Code.org, 2016, Coleman et al., 2016; Lavigne et al., 2020), few of those have undergone a process of development and testing informed by a theoretical framework. In contrast, the CAL curriculum followed the phases described by Clements (2007). Although Clements' work focused on early mathematics curriculum, his CRF can be applied to other disciplines such as computer science (Clements, 2008; Clements & Sarama, 2011).

CRF proposes a 10-phase process of curriculum design to warrant the claim that a curriculum is based on research (Clements, 2007). The first of these phases is the process of identifying subject-matter content that is valid within the discipline of choice (Tyler, 1949). This stage of CRF was CAL's major contribution given that CS education is a relatively new field in early childhood education. CRF's second research phase identifies philosophies that offer new perspectives on students' and teachers' experiences with curricula. In CAL, this involved identifying ways of knowing that are specific for CS, such as learning by designing and programming, as well as perspectives that integrate socio-emotional growth alongside technical skills. CRF's third phase identifies an a priori pedagogical foundation. In CAL, the pedagogical foundation is the understanding of coding as a literacy and draws from the extensive knowledge gained over the years about successful literacy instruction in early childhood. CRF's fourth phase is focused on structuring activities in accordance with domain-specific models of learning (Clements & Battista, 2000; Clements et al., 1992). In the domain of early childhood computer science, these activities involve playfulness, project-based team work, coding and unplugged experiences. The fifth phase presented by CRF is market research at several points in the development cycle. The remaining five phases span both formative research in small groups, single classrooms and multiple classrooms, and summative research in small and large scale. This paper will present results from a pilot study and a cluster randomized controlled trial focusing on two computer science learning outcomes: coding and computational thinking.

2.2. Constructionism

Constructionism, developed by Papert (1980), has its name as a play on Piaget's Constructivism to highlight the importance of learning by doing (Ackermann, 2001; Papert & Harel, 1991). While Piaget's work explains how knowledge is constructed in our heads through a process of accommodation and assimilation, Papert pays particular attention to the role of computationally rich constructions in the world. Programming is both a vehicle for creating meaningful projects and for developing powerful ideas such as sequencing, abstraction, and modularity (Bers, 2008).

Constructionism is consistent with the general agreement in early childhood education about the efficacy of "learning by doing" and engaging in project-based learning (Diffily & Sassman, 2002; Krajcik & Blumenfeld, 2006). It extends these approaches to also engage children in "learning by designing" and "learning by programming" and emphasizes interest-driven and peer-supported activities (Papert, 1993, Kafai & Burke, 2014, Kafai, 1994). When taken into the early childhood classroom, Constructionism supports the creation of coding playgrounds (Bers, 2012; 2018; 2020b; 2022) in which students can gain deeper understandings through the process of coding, reflecting and explaining their own creations.

Starting with LOGO, the first programming language specifically created for children and then expanding to various coding platforms such as Scratch, ScratchJr and KIBO, constructionist programming environments are designed to be open-ended with a "low floor, high ceiling" approach so children can become creative producers instead of consumers of technology (Bers, 2020b; Bers, 2021; Feurzeig et al., 1970; Papert & Harel, 1991; Resnick et al., 2009).

In addition, these coding playgrounds must also engage children in discovering powerful ideas. In his seminal work "Mindstorms: Children, Computers and Powerful Ideas," Papert (1980) coined the term "powerful ideas" to refer to a central concept within a domain that is at once personally useful, epistemologically interconnected with other disciplines, and rooted in intuitive knowledge that a child has internalized over a long period of time (Bers, 2017).

According to Papert, powerful ideas afford new ways of thinking, putting knowledge to use, and making personal and epistemological connections with other domains of knowledge (Papert, 2000). Over the years, a growing community of researchers and educators have used the term powerful ideas to refer to sets of intellectual tools identified as important by a community of experts in each of the fields of study (Bers, 2008; Resnick, 2021). Although programming languages are changing rapidly, the powerful ideas of CS are consistent over long periods of time. The CAL curriculum is organized around seven developmentally appropriate powerful ideas from the field of computer science aligned with powerful ideas from literacy.

2.3. Positive technological development

PTD is a natural extension of Constructionism, but it explicitly incorporates psychosocial, civic, and ethical components. PTD builds on the tradition of Positive Youth Development (PYD), which looks at pathways or developmental assets of thriving individuals in the first two decades of their lives and identifies positive characteristics (Benson et al., 2006; Damon, 2004; Lerner, Almerigi, Theokas, & Lerner, 2005). PTD extends this work to technologically rich interventions, such as the coding playground. However, instead of emphasizing developmental assets, PTD focuses on behaviors: content creation, creativity, collaboration, communication, community building, and choices of conduct (Fig. 1). Programming languages, like many other technologies, empower individuals to do things, to engage in activities, and to act within a community. As shown in Fig. 1, there is a bidirectional relationship between developmental assets and technology-supported behaviors within a particular sociocultural context.

The PTD framework is inspired by the question: "How can we live a purposeful life?" and extends the question to the domain of learning technologies by asking "How can we use new technologies in purposeful ways to become better versions of who we are and create better communities?" CAL answers this question by offering a curriculum with a focus beyond CS mastery and technical skills that also supports children in their quest to develop a sense of identity, values, and purpose within a community (Bers, 2012, 2022).

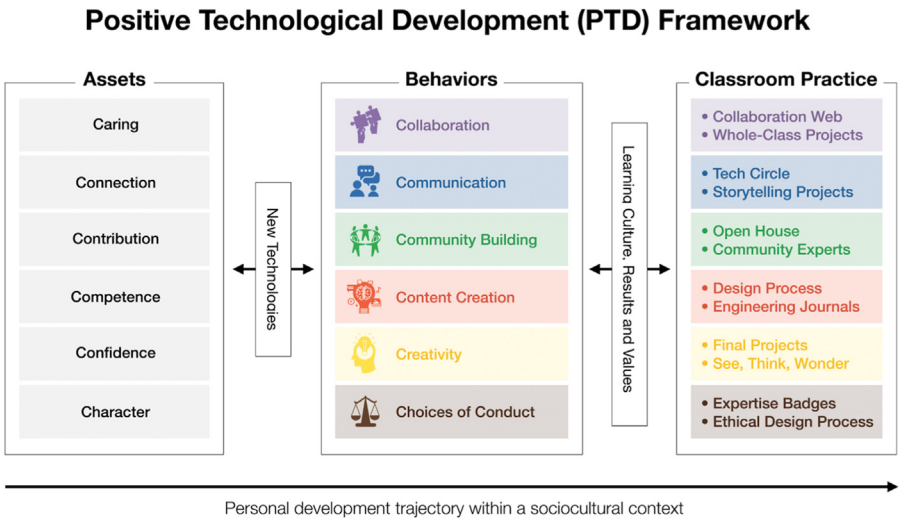

Fig. 1. Positive technological development framework the PTD framework showing the bidirectional relationship between developmental assets and technology-supported behaviors. reprinted from coding as a playground: programming and computational thinking in the early childhood classroom, second edition (p.131), by M. U. Bers, 2020a, Routledge press. copyright 2020 by Routledge press.

3. Pedagogical premise: coding as a literacy

CAL's pedagogical premise is that coding is a literacy for the 21st century. As a literacy, coding is perceived as involving a set of skills and knowledge that today's society highly values (Vee, 2017). While other domains such health literacy, cultural literacy, and visual literacy have also come to be understood as literacies, alphabetical literacy is a unique historical and social phenomenon with strong epistemological implications. It restructures the way we know the world and the way we think (Ong, 1982). The CAL curriculum builds on this function of alphabetical literacy.

CAL supports the exploration of similarities and differences between natural and artificial languages during the creation of computational projects (Ivanova et al., 2020; Fedorenko et al., 2019) and applies strategies for teaching alphabetical literacy to the coding playground (Bers, 2019b). The goal of literacy, as addressed in CAL, is not only for children to master the syntax and grammar of language, but also the meanings and uses of words, sentences, and genres. CAL puts problem solving at the service of personal expression, just as best practices in literacy education place decoding and writing skills at the service of student meaning-making and self-expression (Shanahan & Lonigan, 2013). A literate person knows that reading and writing are tools for meaning making and, ultimately, tools of power. The same is true of coding (Bers, 2020a; Govind et al., 2021).

CAL proposes that programming, as a literacy, engages new ways of thinking, communicating, and expressing ideas. Thus, within the CAL pedagogy, learning to program is akin to learning how to use a symbolic system, such as a written language, to generate and communicate ideas by making a shareable product that can travel away from the author and be read and interpreted by others (Bers, 2018).

Developing alphabetical literacy involves a progression of skills beginning with the ability to understand spoken words, followed by the capacity to code and decode written words, and culminating in the deep understanding, interpretation, and production of text (Chall, 1983). The assumption is that literacy is not a naturally developing process like speech, which unfolds in a child given the right conditions, but requires appropriate instruction (Goldenberg, 2013). The same applies to learning to code. The coding progression does not just happen naturally, but requires appropriate instructional strategies applied to the learning of a new domain, computer science, and the identification of its most powerful ideas that are developmentally appropriate.

4. Powerful ideas of CAL

Three phases were involved in selecting CAL's developmentally appropriate powerful ideas: first, identifying powerful ideas in the field of CS; second, evaluating if those ideas were developmentally appropriate for young children; and third, selecting powerful ideas from other disciplines taught in early childhood that could naturally intersect with CS, most specifically with math (Flannery et al., 2013) and with literacy (Hassenfeld & Bers, 2020; Hassenfeld et al., 2020). The following table (Table 1) summarizes powerful ideas that could be integrated across kindergarten, first grade and second grade.

5. The CAL curriculum

The term "curriculum" has different meanings (Beauchamp, 1986; Jackson, 1992; Pinar et al., 1995; Walker, 2003). Whereas some define curriculum as the totality of learning experiences provided by an educational institution that reflect the society's broader social and political goals, others narrow the definition to focus on a particular course of study. In this paper, the term refers to a written scope and sequence of instructional activities within the domain of computer science using a particular programming language, ScratchJr (Flannery et al., 2013; Kazakoff & Bers, 2013; Strawhacker et al., 2015; Strawhacker et al., 2015; Portelance, Strawhacker, & Bers, 2015).

To determine the scope and sequence of CAL's instructional activities, design-based research was used (Brown, 1992; Cobb et al., 2003; Collins, Joseph, & Bielaczyc, 2004; Design-Based Research Collective, 2003). This process involved the development, field testing, and revision of each of the units in the curriculum, with this cycle repeated several times, and the grounding on appropriate theoretical frameworks. (Bannan-Ritland, 2003; Cunningham et al., 2020).

Once the first version of CAL was developed, we shared it with a 28-member cohort of educators and practitioners participating in a year-long graduate certificate program focused on early childhood technology (Strawhacker et al., 2022). These graduate-level students reviewed CAL and shared feedback on the scope and sequence, lesson activities, and feasibility of implementation, all of which was incorporated in curriculum revisions.

Later, we conducted studies involving hundreds of children, administrators, curriculum coordinators, and technology specialists from multiple schools. We administered and analyzed pre- and post-instructional surveys completed by educators implementing CAL activities in their classrooms, and we reviewed student work and coding projects produced by the children (Leidl, Bers, & Mihm, 2017; Unahalekhaka, & Bers, 2022) We conducted expert review of materials for content, pedagogy, and accuracy to evaluate alignment of standards. We explored how different options for wording and timing of the activities worked and which prompting and scaffolding strategies were most successful. Furthermore, with a lens towards equity (Confrey, 2000; NCTM, 2000), we made inclusive choices of books and songs to be integrated in CAL.

Table 1
Powerful ideas.

Powerful Ideas Computer Science	Powerful Ideas Literacy	Powerful Ideas Mathematics	Connecting the Powerful Ideas
Algorithms	Sequencing	Counting and Patterns	Emphasis on "order matters" and on how complex tasks can be broken down into step-by-step instructions in a logical way.
Design Process	Writing Process	Problem Solving	Creative, iterative, cyclic processes that involve imagining, planning, revising, and sharing, often with different starting points.
Representation	Alphabet and Letter-Sound Correspondence	Cardinality and Interpreting Charts/Graphs	Sounds and symbols have different attributes that can be used to represent something else (e.g., a letter or quantity).
Debugging	Editing and Audience Awareness	Error Checking and Problem Analysis	Systematic analysis, testing, and evaluation to get to the right outcome. Whenever miscommunication occurs, the individual uses a variety of strategies to solve the problem.
Control Structures	Literary Devices	Order of Operations	Advanced strategies that determine how a set of ideas or commands are executed.
Modularity	Phonological Awareness	Place Value	Decomposition, or breaking down a complex task into smaller tasks and re-using those new modules in different ways.
Hardware/Software	Tools of Communication and Language	Tools for Measurement and Computation	Communicating abstract ideas through tangible means.

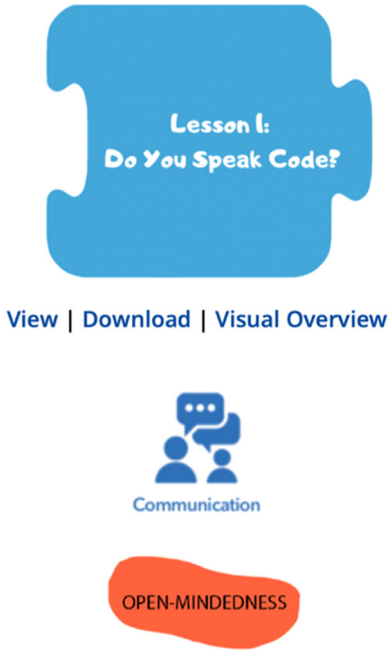

Fig. 2. Elements of PTD and the palette of virtues emphasized in a lesson of the CAL curriculum. Each lesson of the CAL curriculum highlights elements of the positive technological development framework and the palette of virtues metaphor that can be emphasized throughout the lesson. In this figure, Lesson 1 of the first-grade curriculum highlights both communication and open-mindedness

These early formative experiences allowed us to solidify CAL's scope and sequence of both coding and unplugged activities, as well as develop better interdisciplinary integrations. Our research with teachers made evident that, while integration with multiple domains was desired, alphabetical literacy had to be strongly prioritized in the early childhood setting. In addition, we heard from teachers that, for most of them, the CAL curriculum did a good job at integrating the teaching of coding skills with literacy, but they needed more pedagogical support to strengthen socio emotional learning in the coding playground. Based on this feedback, we revised each lesson to add activities to promote and highlight the different C's of the PTD framework and the values highlighted in the palette of virtues (Bers, 2022). These pedagogical elements are now indicated in the top section of the curriculum documents as well as shown through icons on each of the lessons on the website (Fig. 2).

Based on teachers' feedback, we modified some of our original book choices. For example, teachers indicated that the original first-grade nonfiction book we selected, Ada Lovelace: Poet of Science, was too advanced for first grade students. We replaced it with another book about the same female computer scientist: Ada Byron Lovelace and the Thinking Machine. Finally, teachers expressed the need to support all students in their classrooms, including those who might be gifted or who tend to finish their work earlier than others. Thus, we created a section for differentiated learning with proposed projects or challenges for the advanced students.

The resulting CAL curriculum (Fig. 3) has 24 lessons designed for a total of 18 hours, each lesson consisting of warm up and unplugged activities, structured and expressive ScratchJr explorations, and journaling and discussion activities that strengthen the literacy connection, such as word time. The CAL curriculum is free and publicly available on this website: https://sites.bc.edu/codingasanotherlanguage/.

CAL embeds formative and summative assessments. There are six "Checks for Understanding" so teachers can evaluate students' knowledge and skills in a formative way. Based on the computational concepts taught up to that point in the curriculum, these are a series of fun questions that teachers can ask their students as a group. Children respond with a thumbs up for "yes" or a thumbs down for "no." Depending on the responses, teachers can stop and re-explain concepts and skills as needed. For example, in Lesson 8 for kindergarten, children learn about parameters. The Check for Understanding shows two programs side by side: 1) move right, move right, move up; and 2) move right two, move up. The question is: "Will the programs make Cat do the same thing?" If the child has a grasp on how parameters work in ScratchJr, they will respond positively. If not, this will indicate to the teacher that parameters should be reviewed again before moving forward with the curriculum.

For summative evaluation, CAL uses the Show What You Know (SWYK) tool (Fig. 5). SWYK are designed to be run synchronously in class at the beginning of Lesson 22 for every grade. Each child receives a paper answer booklet, and the teacher presents slides with images and videos while they read out loud the questions to the class. The teacher gives the class about two minutes for each question before moving on. The questions begin with the most basic concepts, such as sequencing of movement blocks, and progress to more complex concepts such as identifying specific blocks within a repeat loop or choosing which trigger block should be used for a certain context. The SWYK assessment is mapped closely to the curriculum, meaning each grade has a different version of SWYK to match the different curriculums. This assessment consists of 10 multiple choice questions, and students receive 1 point per correct answer for a total score between 0 and 10.

6. CAL in action: studies description

After the CAL curriculum was completed using the iterative design-based process described earlier, we set up to conduct research to un-

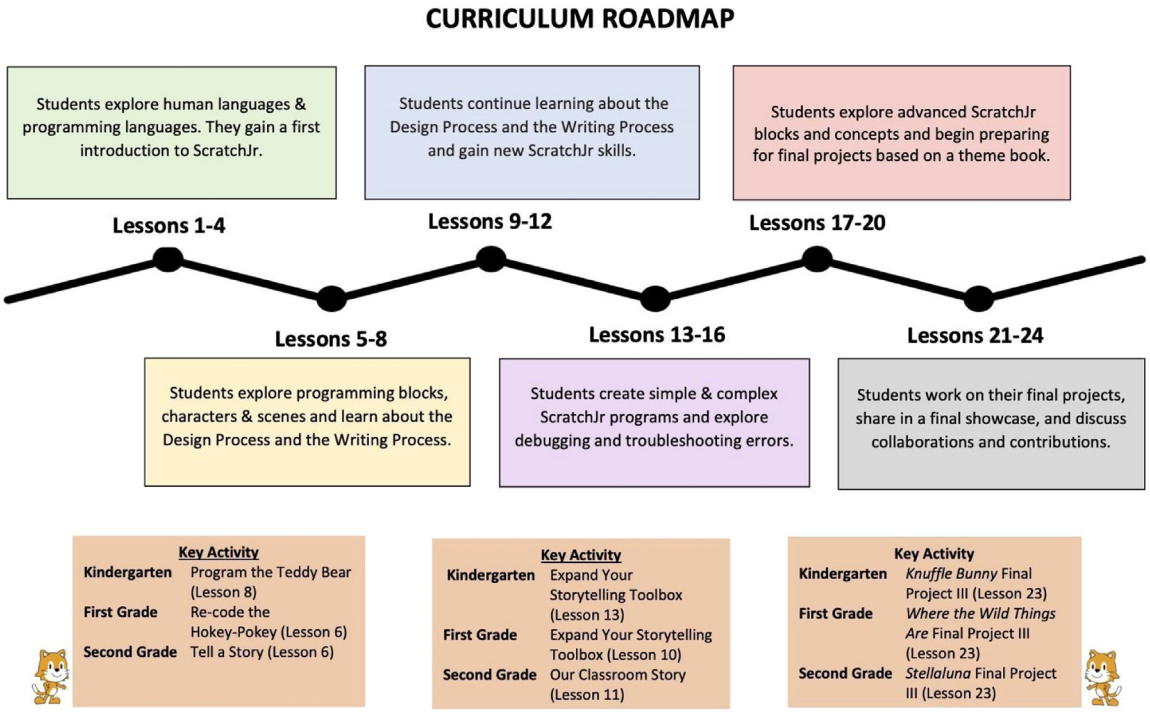

Fig. 3. Curriculum roadmap note. After launching the CAL website, we learned it was hard for teachers to understand the overall flow of the powerful ideas and curricular activities in CAL, without reading the complete document. Thus, we added lesson arcs in the form of "visual roadmaps" for the kindergarten, first, and second grade curricula, as well as for each individual lesson, allowing teachers to have a quick and easy option for reviewing materials and powerful ideas in addition to reading through the entire lesson plan (Figure 4).

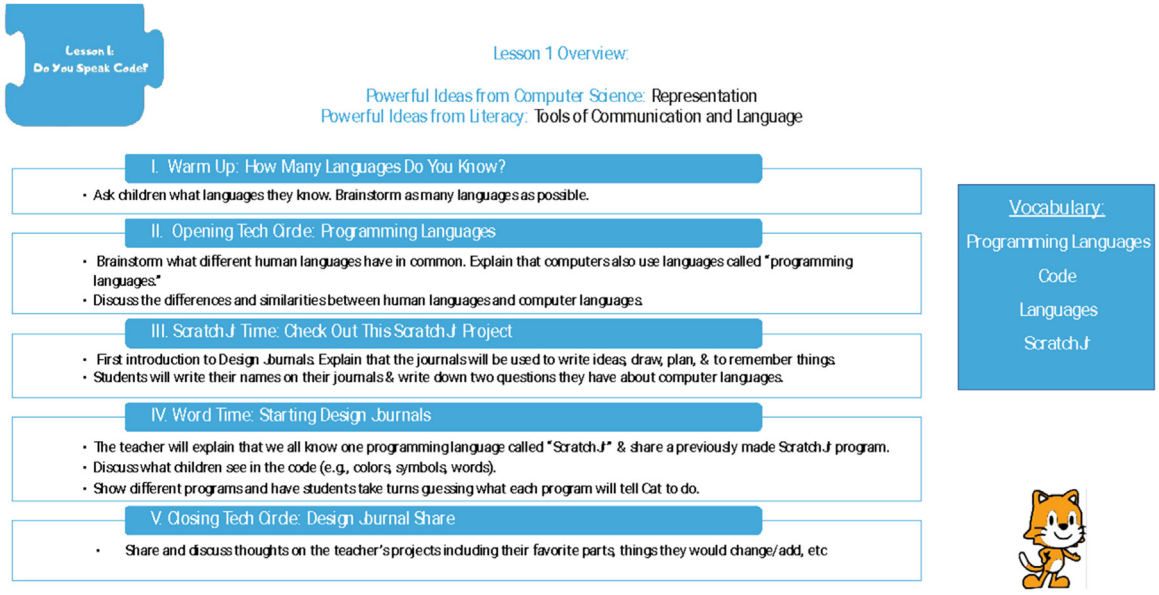

Fig. 4. Visual Roadmap for a lesson within the CAL curriculum Note. This figure shows a visual overview of Lesson 1, which was created as a resource for teachers to understand the flow of the lesson.

derstand whether CAL is feasible to implement and effective in helping children develop coding skills and computational thinking. First, we conducted a pilot study in classrooms in three different states to evaluate the feasibility of teaching CAL in different contexts and of administering our research instruments. While analyzing this preliminary data, we also explored whether the time spent on CAL, which took away from the teaching of literacy and math, had an impact on language and math outcomes. In the pilot phase, no control classrooms were involved. Second, we conducted a cluster-randomized controlled trial through a partnership with the Rhode Island Department of Education.

6.1. Methods

In this article, we report results of a pilot study and a randomized control trial (RCT) to evaluate the feasibility and learning outcomes of CAL curriculum. The pilot and the RCT studies share the same method-

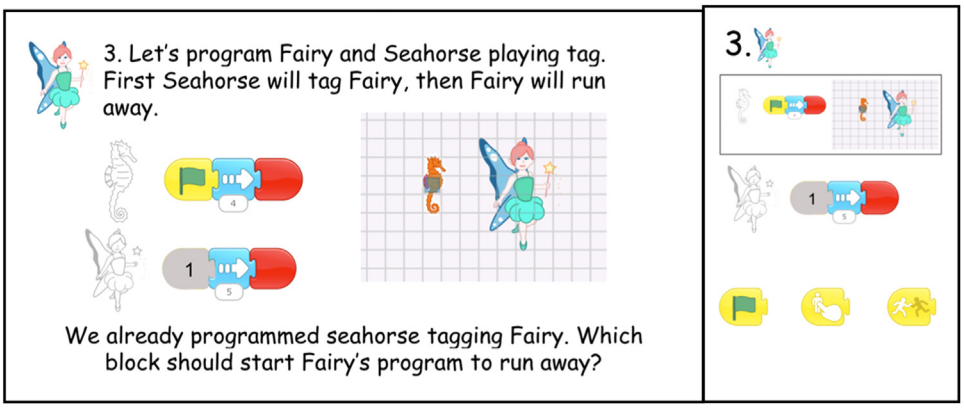

Fig. 5. An Example of show what you know (SWYK) Note. Question 3 of the show what you know (SWYK) assessment presents a scenario to the class in which a seahorse is programmed to move towards a fairy. The question asks "Which block should start Fairy's program to run away?" and presents three options: the Start on Flag, Start on Tap, and Start on Bump blocks.

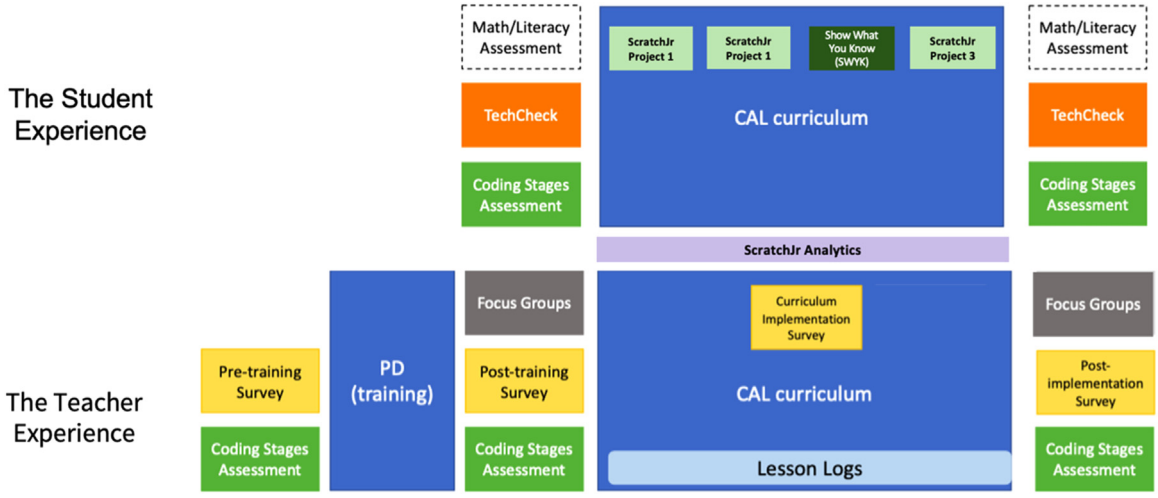

Fig. 6. An Overview of the CAL research study note. Overview of all research elements in the CAL Research study, divided by teacher and student experience.

ological procedures. However, the RCT design included a control group. In this methods section, we first present the shared procedures and common measures by the two studies and then methods for each study including participants and specific instruments.

Fig. 6 provides an overview of the research activities around the intervention. The top panel shows the assessments that participating students completed before, during, and after the experience with the CAL-ScratchJr curriculum from left to right. The large block indicates the period of curriculum implementation, and the small rectangles represent assessments and surveys. Students completed two instruments before and after participating in the CAL curriculum: TechCheck, which assesses computational thinking, and Coding Stages Assessment (CSA), which evaluates coding knowledge with ScratchJr. The CSA is administered one-on-one between researcher and participant. The researcher displays either images or videos to accompany each question. The participant answers each question on a tablet, and the researcher then indicates the answer as satisfactory or unsatisfactory. We administered both CSA and TechCheck assessment virtually over a video chat platform (Zoom) to accommodate for COVID-19 safety precautions. The instruments are described in detail later and shown at their collection points for teachers and students.

While this article does not focus on teacher outcomes, to describe the overall experience, Fig. 6 also includes teacher activities. The lower panel of this figure shows the teachers' experience. Teachers first completed the pre-training survey and the CSA, and then they attended a 4-hour professional development (PD) training with the DevTech Research Group. The PD training involved guided explorations and open-ended play with the ScratchJr app, a deep-dive into the CAL curriculum and an introduction to the pedagogical approach for making the classroom into a coding playground (Bers, 2020a). After the training, teachers filled out a post-training survey and completed the CSA again and some teachers opted in to do a focus group interview. The teaching of the CAL curriculum spanned 12 weeks, comprising a total of 24 sessions of 30-45 minutes each. During the CAL-ScratchJr curriculum experience, teachers filled out curriculum implementation surveys and lesson logs; after the curriculum implementation, once again, teachers completed out the post-implementation survey, CSA, and focus group interview. Results from this experience have been previously shared (Kapoor et al., 2023). In this paper we report in coding and computational thinking student learning outcomes.

6.2. Measures

The Coding Stages Assessment (CSA). The CSA was used to understand individual learning trajectories regarding coding knowledge and skills using ScratchJr. CSA is a validated assessment that has a series of open-ended task-based questions about ScratchJr (de Ruiter & Bers, 2021). It is constructed around five coding stages: Emergent, Coding and Decoding, Fluency, New Knowledge, and Purposefulness (Bers, 2019a). Each of the coding stages has six questions that probe the specific skills and thought processes that define the coding stage. For example, questions targeting the emergent coding stage focus on interface comprehension and symbolic representation whereas the questions targeting the fluency stage ask participants to produce programs that utilize complex blocks such as messages and trigger blocks. The first four stages progress linearly through levels of difficulty and complex-

Table 2
Pre- and post- curriculum scores for coding skills and computational thinking.

Assessment	N	Baseline Score (M ± SD)	End Point Score (M ± SD)	Points Changed(M ± SD)	Paired Sample t-test p value
Weighted CSA[a]	166	6.5 (4.9)	17 (7.5)	10.7 (7.2)	P<.0001
TechCheck[b]	163	8.53 (2.67)	9.85 (2.69)	1.31 (2.28)	P<.0001

Note. CSA[a]: Coding Stages Assessment measures ScratchJr coding skills
TechCheck[b]: Unplugged assessment for computational thinking

Table 3
Descriptive statistics of pre- and post-curriculum CSA scores from three grade levels.

	CSA Scores							
	Min.		Max.		Mean		SD	
	Pre	Post	Pre	Post	Pre	Post	Pre	Post
Kindergarten	1.1	4.4	11.7	20.9	5.4	13	3	4.6
1st Grade	0	3.3	22	29	5	14.9	3.8	5.8
2nd Grade	1.1	0	29.1	39	9	20.8	5.6	8.5
K-2	0	0	29.1	39	6.5	17	4.9	7.5

ity. If the participant answers five out of six correctly, she will move on to the next stage. If not, the assessment will end. To accommodate the adaptive nature of the CSA, a weighted score is calculated. Specifically, the number of questions correctly answered for each stage are totaled and differentially weighted, with weights ranging from 1.1 for Emergent stage questions to 1.5 for Purposefulness questions. Possible weighted CSA scores range from 0 to 39.

To further ensure inter-rater reliability, 20% of students were randomly assigned a second rater, andthe data in this study indicated a substantial interrater reliability (Cohen's $\kappa = 0.85$).

TechCheck. In addition to the CSA, children completed TechCheck to assess computational thinking skills before and after the CAL intervention. TechCheck is a 15-item, multiple-choice assessment that probes six domains of CT described by Bers' (2018) as developmentally appropriate for young children (algorithms, modularity, control structures, representation, hardware/software, and debugging). This assessment presents children with puzzle-like challenges and it does not require the child to have programming knowledge or computing experience. Items are scored as correct or incorrect, and the number of correct answers is totaled with possible scores ranging from 0 to 15. The assessment can be administered in several modalities – online or in person, and to individuals, whole classrooms, or groups of students – and can be administered in 20 minutes or less. TechCheck has been validated and used both cross-sectionally and longitudinally with kindergarten, first, second, and third grade students between five and nine years old (Relkin et al., 2021; Relkin & Bers 2021; Relkin et al., 2020; Relkin et al, 2023).

6.3. Pilot Study

Participants. During the pilot study, the CAL curriculum was evaluated in four schools: two in San Francisco, one in Minnesota, and one in Arkansas, reaching a total of 224 students. Even though demographics such as socioeconomic status (SES) were not provided from each individual student by the schools, these school sites in this pilot study represent a diverse student population. For example, 71.9% of the students in Arkansas, 53.1% of the students in San Francisco, and 19% of the students in Minnesota were eligible to participate in the federal free and reduced-price meal program. We purposefully chose to run a pilot study in locations with different populations to understand feasibility of CAL implementation and administration of research instruments.

Math and Literacy Assessments. In addition to the CSA and TechCheck, in the pilot study, we obtained three different types of standardized literacy and math assessments from two different time points: Winter (before the CAL curriculum was taught) and Spring (after the CAL curriculum was taught). Specifically, Measure of Academic Progress (MAP) literacy and math assessments were obtained from first grade participants in Arkansas. FastBridge reading and math assessment scores were obtained from first grade students in Minnesota. STAR Reading and Math assessment scores were collected from second grade students in Minnesota.

6.4. Results of pilot study: student learning outcomes

Over the course of CAL implementation, both students' coding skills and their computational thinking showed significant growth (Table 2). The TechCheck assessment, measuring computational thinking, was taken by students before and after the CAL intervention. The TechCheck sample consisted of 173 total students: 12 kindergarten students, 88 first grade students, and 63 second grade students. Overall, the mean score at baseline was 8.53 (SD= 2.67), and the mean score following the curriculum was 9.85 (SD= 2.69). This change was highly significant according to a paired sample t-test ($t(162) = 5.02$, $p < 0.001$).

The CSA, measuring coding skills, was taken by a total of 188 students, of which 166 students completed both the pre-curriculum and post-curriculum assessments. The sample consisted of 15 kindergarten students, 87 first grade students, and 64 second grade students. The mean weighted score at baseline was 6.5 (SD= 4.9), and the mean weighted score following the curriculum was 17.0 (SD= 7.5) (27.43% increase out of a total 39 points), which was a highly significant change according to paired sample t-test, $t(165) = 19.05$, $p < 0.0001$.

Table 3 presents descriptive statistics (minimum, maximum, mean, and standard deviation) of the pre-curriculum and post-curriculum weighted CSA scores from kindergarten, first, and second grade students.

Students from all grade levels had a significant increase in their CSA scores following the curriculum implementation (Table 3). The paired sample t-test statistics for kindergarten, first grade, and second grade are $t(14) = 5.02$, $p < .001$, $t(86) = 14.91$, $p < .001$, $t(63) = 10.41$, $p < .001$, respectively. ANOVA analysis showed the change of score between pre-curriculum and post-curriculum CSA is significantly different across grades, $F(2, 163) = 4.04$, $p < .05$. All grade levels had significant increase of CSA after the curriculum implementation, but estimation of a Games-Howell post-hoc test revealed that second grade students had significantly higher improvement of CSA when compared to kindergarten students ($d = 4.9$, $p < .05$). It is not clear whether the between-grade differences reflect age-related changes in coding performance or differences between grades in the psychometric performance of the CSA. Students' CSA growth over the curriculum implementation had a significant positive correlation with TechCheck difference scores over the curriculum implementation ($r(148) = .18$, $p = .03$).

CAL was taught during academic hours that would have been allocated to the math or literacy block. Thus, during the pilot study, we wanted to examine whether time spent learning to code has or does not have a detrimental effect on students' achievements in these other subjects. This finding would determine CAL's feasibility. Thus, standardized literacy and math scores were obtained from a subset of students in two study locations (Arkansas and Minnesota) from two different time points: Winter (before the CAL curriculum was taught) and Spring (after the CAL curriculum was taught).

Table 4
Background characteristics.

Background Characteristics	Control Group (*n*)	% of Control group (*n* = 488)	Treatment Group (*n*)	% of Treatment group (*n* = 464)
Female	251	51.4%	233	50.2%
Limited English Proficiency	33	6.8%	54	11.6%
Individualized Education Plan	60	12.3%	54	11.6%
Free/reduced-price lunch	87	17.8%	83	17.9%

Table 5
Descriptive statistics of pre- and post-curriculum CSA scores from three grade levels.

	Control					Treatment				
	n	Mean		SD		n	Mean		SD	
		Pre	Post	Pre	Post		Pre	Post	Pre	Post
Kindergarten	89	2.31	4.21	1.47	2.43	145	1.88	8.69	1.26	3.82
1st Grade	200	3.10	5.30	1.86	3.38	136	3.16	14.08	1.99	6.81
2nd Grade	164	4.50	7.23	2.71	4.49	149	4.63	15.69	3.40	8.32

Results from the Measure of Academic Progress (MAP) literacy and math assessments were obtained from 59 first grade students in Arkansas. FastBridge reading and math assessment scores were obtained from 48 first grade students in Minnesota. STAR reading and math assessment scores were collected from 41 second grade students in Minnesota.

Wilcoxon signed-rank tests indicate that MAP Rasch Unit (RIT) math and literacy measures showed significant improvement over the course of CAL implementation among first grade students in Arkansas ($V = 1414.5$, $p < .001$; $V = 1368.5$, $p < .001$). FastBridge math and reading scores in the first grade from Minnesota showed significant growth over the course of the curriculum implementation ($t(46) = 8.95$, $p < .001$; $t(46) = 16.26$, $p < .001$), as well as the STAR math and reading scores in the second grade from Minnesota ($t(40) = 16.86$, $p < .001$; $t(40) = 11.29$, $p < -.001$). Overall, there was improvement seen over time. However, to answer if such improvement was due to maturation or implementation of the CAL curriculum would require an experiment including a control group. Although in the pilot study the cause of improvement in the standardized scores is unknown, there is evidence that learning to code during the academic school day had no negative impact on math and reading performance. Following the pilot study, a clusterrandomized controlled trial was conducted.

6.5. Randomized controlled trial study

Participants. 952 students from 13 schools in Rhode Island through partnerships with The Rhode Island Department of Education participated in this research. Thirteen schools were randomly assigned to the intervention or control conditions. Specifically, 98 kindergarten participants from five schools were in the control condition and 158 kindergarten students from four schools were in the intervention condition; 219 first-grade participants from seven schools were in the control condition and 146 first-grade students from five schools were in the intervention condition; and 171 second-grade students from six schools were in the control condition and 160 second-grade students from five schools were in the intervention condition. The schools in the intervention group implemented the CAL curriculum from Fall 2021 to Spring 2022 (approximately 5 months). The control group did business as usual. Both groups were assessed around the same time before and after the curriculum regarding students' coding ability as measured by CSA and their computational thinking ability as measured by TechCheck. Table 4 presents the demographics of the student participants in treatment and control groups. Regarding the demographic equivalence, gender, IEP status, and social-economic status are very similar between the two groups, but for students' limited English proficiency (LEP) status, the treatment group contained a higher percentage of students with limited English proficiency.

Table 6
Fixed effects of treatment on post coding performance (Post-CSA) controlling for covariates.

Fixed effects	B	SE_B	*p* value
Intercept	4.89	2.30	0.06
Condition (Treatment)	7.28	0.83	<0.001
Pre-CSA	0.88	0.07	<0.001
Grade (2nd)	0.35	0.41	0.39
Grade (K)	-3.07	0.48	<0.001
Gender (Male)	0.71	0.33	0.03
IEP (Yes)	-1.64	0.52	0.002
LEP (Yes)	-0.92	0.68	0.17
Free/Reduced Lunch (Yes)	-0.09	0.50	0.86
School level Pre-CSA	-0.35	0.65	0.60
School % of Free/Reduced Lunch	-2.02	2.29	0.40

6.6. Results of RCT study: student learning outcomes

Multi-level modeling was used to analyze the two-level nested data with individual student data serving as the first level and school serving as the second level nesting variable. Before the model specification, baseline equivalence was examined. Both outcomes, coding and computational thinking, met the baseline equivalence between the treatment and the control group. To estimate the impact of the curriculum implementation, fixed effects of the treatment condition were examined at level-one, and random intercept was included at level-two. Finally, fixed effects of student-level baseline score, disability (IEP) status, limited English proficiency (LEP) status, and free/reduced lunch status were included as covariates at level-one and school-level baseline score and percentage of students with free/reduced lunch status were included as covariates at level-two. Table 5 shows the mean and standard deviation of pre and post coding proficiency scores from all grade levels.

According to the unconditional model, students' post CSA shows an ICC of 0.31, which indicates that 31% of the post CSA variance, a very large variance, is due to the between school variability. This confirmed the need of the proposed multilevel modeling adjusting the school clustering effect. After controlling the given covariates, the treatment showed significantly higher post-curriculum CSA scores compared to the control group in all grades studied with an unstandardized coefficient of 6.19 with a large effect size, Hedge's *g* of 0.47, (see table 6). This indicates that when all the covariates were held constant, the first-grade students in the treatment group showed an average of 6.19 point higher increase than the control group in the coding proficiency after the CAL curriculum implementation. The results suggest evidence that students' growth in their coding proficiency is due to the intervention and not maturation.

Table 6.

Table 7
Descriptive statistics of pre- and post-curriculum TechCheck from three grade levels by condition.

	Control					Treatment				
	n	Mean		SD		n	Mean		SD	
		Pre	Post	Pre	Post		Pre	Post	Pre	Post
Kindergarten	87	7.07	7.98	2.39	2.38	141	7.07	8.49	2.33	2.43
1st Grade	195	7.57	9.05	2.37	2.45	132	7.33	8.99	2.57	2.56
2nd Grade	160	7.19	8.89	2.47	2.64	146	6.99	8.60	2.53	2.91

Table 8
Fixed effects of treatment on post-TechCheck controlling for covariates.

Fixed effects	B	SE_B	p value
Intercept	1.77	1.16	0.57
Condition (Treatment)	0.20	0.16	0.10
Pre-TechCheck	0.48	0.03	<0.001
Grade (2nd)	-0.15	0.17	0.35
Grade (K)	-0.63	0.19	0.001
Gender (Male)	0.53	0.15	<0.001
IEP (Yes)	-1.06	0.23	<0.001
LEP (Yes)	-0.12	0.28	0.45
Free/Reduced Lunch (Yes)	-0.13	0.21	0.39
School level Pre-TechCheck	0.51	0.16	0.001
School % of Free/Reduced Lunch	-1.09	0.76	0.18

In addition, given that the focus of this article is on the main treatment impact and not the demographics, the effects of the demographics are not discussed in detail. However, it is worth mentioning that students' coding proficiency did not differ by their LEP status or free/reduced lunch status, but differed by their IEP status. IEP students generally scored lower than their counterparts.

Table 7 presents descriptive statistics of students' pre and post TechCheck scores in each grade by condition. Students showed increases in their computational thinking from baseline scores. According to the unconditional model, students' post-TechCheck showed an ICC of 0.09. This means that 9% of variance is due to the between school variability. As a result of multilevel modeling controlling for all the covariates proposed, TechCheck scores did not show significant difference between the treatment and control group (see Table 8). These results suggest that everyone improved in their computational thinking, but students in the treatment group did not differ significantly compared to their counterparts in the control group, interrogating the similarities and differences between coding and computational thinking processes.

7. Discussion

Early childhood education is currently changing to encompass the teaching and learning of new domains of knowledge that are relevant to today's society, such as computer science and computational thinking. Policies and frameworks are slowly being adapted to mandate or strongly suggest their teaching in the early childhood classroom. With this effort, there is a need for research-based curricula that not only include a structured scope and sequence but that also identify powerful ideas from these domains that are developmentally appropriate for young children. The work presented in this paper describes such an endeavor: the Coding as Another Language (CAL) curriculum for the free and widely available introductory programming language, ScratchJr.

In this paper, we described the process of iteratively developing and testing CAL by building on three theoretical foundations: Curriculum Research Framework, which provided a roadmap for steps to follow to make CAL into an evidence-based curriculum; Constructionism, which oriented the scope and sequence of project-based activities to be designed upon powerful ideas; and Positive Technological Development, which integrated socio-emotional and ethical dimensions. CAL's pedagogical premise is that coding is another language and therefore can be taught by integrating powerful ideas of literacy with computer science.

In addition to these pedagogical foundations, this paper presents results from a pilot study and a cluster randomized controlled trial to evaluate CAL's feasibility in the classroom and its efficacy in meeting its goal to teach coding and computational thinking. Results from the pilot study indicate that the CAL curriculum was not only feasible to implement, but also effective. Furthermore, despite the time taken away from the math and literacy blocks to teach computer science, standardized test scores for literacy improved and scores for math remained the same.

During the cluster-randomized controlled trial, the treatment group's growth in coding skills surpassed the control group at all grade levels. However, both the intervention and control groups improved on computational thinking. This result regarding computational thinking was unexpected and requires further exploration.

There are many plausible explanations for this. First, it is possible that improvements in the control group were a product of typical development and schooling that equaled the effects of the intervention. Past studies have shown that CT skills as measured by TechCheck improve in the course of natural development, even in the absence of coding instruction (Relkin, 2022). Children in this age group change rapidly and it is plausible that in a study that lasted around 5 months between pre and post testing, all children further develop their computational thinking skills. Given that CT is defined as being present in everyday analytical skills (Wing, 2006; 2011), it is possible that some of the constructs of the TechCheck instrument captured what was happening in both the CAL intervention group and business as usual, learning math, literacy or STEM, in the control group. Further research is needed to determine whether students showed greater improvement for some computational thinking constructs than others and to examine what the control classrooms were learning at the time of the study by interviewing teachers in the control group.

A second possible explanation is that the CAL curriculum may not be as effective at increasing CT skills (measured by TechCheck), as it is in teaching children expressive coding activities (measured by CSA). At the heart of computational thinking is abstraction (Kramer, 2007), that is, the ability to identify salient pieces of a problem or model and ignore inessential details. CAL's premise is that coding is another language, and therefore, CAL's aim is to help children express themselves through this new language. The focus is not necessarily on abstraction in problem

solving, but rather on making abstract ideas concrete through the programming language (Papert, 1980) and using and manipulating a symbolic representational system to create a sharable product that others can interpret (Bers, 2020). Thus, when children in the CAL intervention group were given an abstract measure, such as TechCheck, they did not perform significantly better than their peers in the control group. They were not taught how to work with abstraction, but to make abstractions concrete through the use of the ScratchJr language to create meaningful projects. Further investigation regarding this possible explanation is needed by looking at the different items on TechCheck and comparing performance on specific items for students in the treatment and control groups.

8. Conclusion

Computer science education is reaching the younger populations. As schools make room in their busy schedules to include coding projects and unplugged computational thinking activities, there is an urgent need to work with evidence-based curricular resources that have been piloted and evaluated. These need to be both developmentally-appropriate and based on theoretical foundations that position the teaching of computer science in the broader educational discourse. The work on CAL presented in this paper addresses all of these issues. CAL was iteratively designed and tested using three established bodies of literature: Curriculum Research Framework (CRF), which informed the steps to ensure an evidence-based final product, Constructionism, which informed the project-based and designed-based approach to coding in early childhood, and Positive Technological Development, which provided the developmentally appropriate grounding on socio-emotional aspects involved in learning. Furthermore, CAL's design core innovation is the integration of early literacy and computer science by identifying overlapping powerful ideas in a developmentally appropriate way. Finally, this paper provides evidence about CAL's feasibility, and student learning outcomes regarding coding and computational thinking, from both a pilot study and a cluster randomized controlled trial which interrogated the relationship between computational thinking and coding.

As more work is done, the relationship between coding and computational thinking in early childhood needs to be further explored, not only for better discerning the impact of CAL, but also for achieving equity in education. As shown in the work presented here, computational thinking and learning how to use a programming language for expressive purposes are not the same. However, both are important and need to be better understood. As children grow, there is a demand for a technically-savvy workforce to fulfill the needs of the economy. The pipeline starts early on, and those children who are not exposed to coding at an early age might face hardships later. The literature shows that by third grade, stereotypes regarding STEM start to form, and girls and minorities are the most impacted.

The current growing trend showing that computational thinking can be developed in unplugged ways through many disciplinary connections, might also hide the fact that coding is not the same as computational thinking. And while thinking is important, learning how to express that thinking through a programming language is a valued skill. The work reported in this paper can help illuminate other's endeavors to create and evaluate research-based curriculum in this nascent field of early childhood computer science.

Data availability

The authors do not have permission to share data.

Acknowledgment

Funding: This work was supported by the United States Department of Education [Grant No U411C220202].

References

Ackermann, E. (2001). Piaget's constructivism, papert's constructionism: What's the difference. *Future of Learning Group Publication, 5*(3), 438–448.
Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.
Bannan-Ritland, B. (2003). The role of design in research: The integrative learning design framework. *Educational Researcher, 32*(1), 21–24.
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.
Beauchamp, G. A. (1986). Curriculum theory: Meaning, development, and use. *Theory Into Practice, 21*(1), 23–27.
Benson, P. L., Scales, P. C., Hamilton, S. F., Sesma, A., Jr., Lerner, R. M., & Damon, W. (2006). Positive youth development: Theory, research, and applications. In *Handbook of child psychology: Theoretical models of human development* (pp. 894–941). John Wiley & Sons Inc.
Bers, M. (2020a). *Coding as a playground: Programming and computational thinking in the early childhood classroom, second edition*. New York, NY: Routledge Press.
Bers, M. U. (2020b). Playgrounds and microworlds: Learning to code in early childhood. Designing constructionist futures: *The art, theory and practice of learning designs*.
Bers, M., & Bers, M. U. (2021). From computational thinking to computational doing. *Teaching computational thinking and coding to young children*. IGI Global. 10.4018/978-1-7998-7308-2.
Bers, M. U. (2008). *Blocks to robots learning with technology in the early childhood classroom*. New York, NY: Teachers College Press.
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*.
Bers, M. U. (2018). Coding and computational thinking in early childhood: the impact of ScratchJr in Europe. *European Journal of STEM Education, 3*(3), 08.
Bers, M. U., & Donohue, C. (2019a). Coding as another language. In *Exploring key issues in early childhood and technology: Evolving perspectives and innovative approaches* (pp. 63–70). New York, NY: Routledge.
Bers, M. U. (2019b). 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. (2022). *Beyond Coding: How Children Learn Human Values through Programming*. Cambridge, MA: MIT Press.
Bers, M., Govind, M., & Relkin, E. (2021). Coding as another language: Computational thinking, robotics and literacy in first and second grade. *ACM Special Issue on K-5 Computational Thinking*.
Bers, M. U., & Resnick, M. (2015). *The official ScratchJr book*. San Francisco, CA: No Starch Press.
Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions. *Journal of the Learning Sciences, 2*(2), 141–178.
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).
Chall, J. S. (1983). Literacy: Trends and explanations. *Educational Researcher, 12*(9), 3–8.
Clements, D. H. (2007). Curriculum research: Toward a framework for research-based curricula. *Journal for research in mathematics education, 38*(1), 35–70.
Clements, D. H. (2008). *Handbook of design research methods in education: Innovations in science, technology, engineering, and mathematics learning and teaching,* (pp. 410–422). 20 Design Experiments and Curriculum Research.
Clements, D. H., & Battista, M. T. (2000). *Handbook of research design in mathematics and science education* (pp. 761–776). Designing effective software.
Clements, D. H., & Sarama, J. (2011). Early childhood mathematics intervention. *Science, 333*(6045), 968–970.
Clements, D. H., Meredith, J. S., Battista, M. T., Geeslin, W., & Graham, K. (1992). Design of a Logo environment for elementary geometry. In *Proceedings of the sixteenth annual meeting of the North American chapter of the international group for the psychology of mathematics education: 1* (p. 152). Durham, NH: Program Committee.
Cobb, P., Confrey, J., DiSessa, A., Lehrer, R., & Schauble, L. (2003). Design experiments in educational research. *Educational Researcher, 32*(1), 9–13.
Code.org. (2016). Evaluation summary report 2015-2016 https://code.org/files/EvaluationReport2015-16.pdf.
Alliance. (2021). 2021 State of computer science education: Accelerating action through advocacy. Code.org, CSTA, & ECEP. https://advocacy.code.org/stateofcs.
Coleman, L. O., Gibson, P., Cotten, S. R., Howell-Moroney, M., & Stringer, K. (2016). Integrating computing across the curriculum: The impact of internal barriers and training intensity on computer integration in the elementary school classroom. *Journal of Educational Computing Research, 54*(2). 10.1177/0735633115616645.
Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design research: Theoretical and methodological issues. *The Journal of the Learning Sciences, 13*(1), 15–42.
Confrey, J. (2000). Improving research and systemic reform toward equity and quality. In *Handbook of research design in mathematics and science education* (pp. 87–106). Mahwah, NJ: Lawrence Erlbaum Associates.
Cunningham, C. M., Lachapelle, C. P., Brennan, R. T., Kelly, G. J., Tunis, C., & Gentry, C. A. (2020). The impact of engineering curriculum design principles on elementary students' engineering and science learning. *Journal of Research in Science Teaching, 57*(3), 423–453.
Damon, W. (2004). What is positive youth development? *The Annals of the American Academy of Political and Social Science, 591*(1), 13–24.
de Ruiter, L. E., & Bers, M. U. (2021). The coding stages assessment: development and validation of an instrument for assessing young children's profi-

ciency in the ScratchJr programming language. *Computer Science Education*, 1–30. 10.1080/08993408.2021.1956216.
Design-Based Research Collective. (2003). Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher, 32*(1), 5–8.
Diffily, D., & Sassman, C. (2002). *Project-based learning with young children*. Westport, CT: Heinemann, Greenwood Publishing Group, Inc.
Fedorenko, E., Ivanova, A., Dhamala, R., & Bers, M. U. (2019). The language of programming: a cognitive perspective. *Trends in Cognitive Sciences, 23*(7), 525–528.
Feurzeig, W., Papert, S., Bloom, M., Grant, R., & Solomon, C. (1970). Programming-languages as a conceptual framework for teaching mathematics. *ACM SIGCUE Outlook, 4*(2), 13–17.
Flannery, L. P., Kazakoff, E. R., Bontá, P., Silverman, B., Bers, M. U., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th international conference on interaction design and children (IDC '13)* (pp. 1–10). New York, NY: ACM. 10.1145/2485760.2485785.
Goldenberg, C. (2013). Unlocking the research on english learners: What we know–and don't yet know–about effective instruction. *American Educator, 37*(2), 4–11.
Govind, M., Hassenfeld, Z., de Ruiter, L., & Bers, M. U. (2021). Rhyme and reason: the connections among coding, computational thinking, and literacy. *Teaching computational thinking and coding to young children*. IGI Global.
Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher, 42*(1), 38–43.
Hassenfeld, Z. R., & Bers, M. U. (2020). Debugging the writing process: Lessons from a comparison of students' coding and writing practices. *The Reading Teacher, 73*(6), 735–746. 10.1002/trtr.1885.
Hassenfeld, Z. R., Govind, M., de Ruiter, L. E., & Bers, M. U. (2020). If you can program, you can write: Learning introductory programming across literacy levels. *Journal of Information Technology Education: Research, 19*, 65–85. 10.28945/4509.
Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'Reilly, U.-M., … Fedorenko, E. (2020). Comprehension of computer code relies primarily on domain-general executive brain regions. *ELife, 9*, Article e58906. 10.7554/eLife.58906.
Jackson, P. (1992). *Handbook of research on curriculum*. Macmillan No. 375.00973 J1321h Ej. 1 003726.
Kafai, Y. B., & Burke, Q. (2014). *Connected code: Why children need to learn programming*. Cambridge, MA: The MIT Press.
Kafai, Y., & Soloway, E. (1994). Computational gifts for the Barney generation. *Communications of the ACM, 37*(9), 19–22.
Kapoor, M. G., Yang, Z., & Bers, M. (2023). Supporting early elementary teachers' coding knowledge and self-efficacy through virtual professional development. *Journal of Technology and Teacher Education, 30*(4), 1–31 2023.
Kazakoff, E. R., Sullivan, A., & Bers, M. U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal, 41*(4), 245–255.
Krajcik, J. S., Blumenfeld, P. C., & Sawyer, R. K. (2006). Project based learning. In *The Cambridge handbook of learning sciences* (pp. 317–334). New York, New York: Cambridge University Press.
Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM, 50*(4), 36–42.
Lavigne, H., Presser, A. L., Rosenfeld, D., Wolsky, M., & Andrews, J. (2020). Creating a preschool computational-thinking learning blueprint to guide the development of learning resources for young children. *Connected Science Learning, 2*(2).
Leidl, K. D., Bers, M. U., & Mihm, C. (2017). Programming with ScratchJr: a review of the first year of user analytics. In *Proceedings of the conference proceedings of international conference on computational thinking education* (pp. 116–121).
Lerner, R. M., Lerner, J. V., Almerigi, J. B., Theokas, C., Phelps, E., Gestsdottir, S., … Von Eye, A. (2005). Positive youth development, participation in community youth development programs, and community contributions of fifth-grade adolescents: Findings from the first wave of the 4-H study of positive youth development. *The Journal of Early Adolescence, 25*(1), 17–71.
Lodi, M., & Martini, S. (2021). Computational thinking, between papert and wing. *Science & Education*, 1–26.
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.
NCTM. (2000). *Standards for school mathematics*. Reston, VA: National Council of Teachers of Mathematics.
Ong, W. (1982). *Orality and literacy: The technologizing of the word*. Routledge.
Paper, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books, Inc.
Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. New York, NY: BasicBooks.
Papert, S. (2000). What's the big idea? toward a pedagogy of idea power. *IBM Systems Journal, 39*(3.4), 720–729.
Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism, 36*(2), 1–11.
Pinar, W. F., Reynolds, W. M., Taubman, P. M., & Slattery, P. (1995). *Understanding curriculum: An introduction to the study of historical and contemporary curriculum discourses*: 17. Peter Lang.
Portelance, D. J., Strawhacker, A., & Bers, M. U. (2015). Constructing the ScratchJr programming language in the early childhood classroom. *International Journal of Technology and Design Education*, 1–16. 10.1007/s10798-015-9325-0.
Relkin E. & Bers M.U. (2021). Factors influencing learning of computational thinking skills in young children. Virtual Annual Meeting of the American Educational Research Association (AERA).
Relkin, E., de Ruiter, L. E., & Bers, M. U. (2021). Learning to code and the acquisition of computational thinking by young children. *Computers & Education*. 10.1016/j.compedu.2021.104222.
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*, 428–498. 10.1007/s10956-020-09831-x.
Relkin, E. V. (2022). *The development of computational thinking skills in young children(Order No. 29326738)*. ProQuest Dissertations & Theses Global (2725323425). Retrieved from https://www.proquest.com/dissertations-theses/development-computational-thinking-skills-young/docview/2725323425/se-2 .
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., Flanagan, M., Kelleher, C., MacLaurin, M., Ohshima, Y., Perlin, K., & Torres, R. (2009). *CHI'09 Extended Abstracts on Human Factors in Computing Systems* (pp. 3293–3296). Growing up programming: democratizing the creation of dynamic, interactive media.
Shanahan, T., & Lonigan, C. J. (2013). *Early childhood literacy: The national early literacy panel and beyond*. Towson: MD: Paul H. Brookes Publishing Company.
Strawhacker, A., Portelance, D., Lee, M., & Bers, M. U. (2015). Designing Tools for developing minds: The role of child development in educational technology. In *Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15)*. New York, NY: ACM Medford, MA, June 21-25.
Sullivan, A., & Bers, M. U. (2019). Computer science education in early childhood: The case of ScratchJr. *Journal of Information Technology Education: Innovations in Practice, 18*, 113–138.
Strawhacker, A., Govind, M., & Bers, M. (2022). Understanding the experiences of early childhood professionals' navigation of remote teaching and learning with technology [Paper presentation]. American Educational Research Association Annual Meeting. San Diego, CA.
Tucker A., Deek F., Jones J., McCowan D., Stephenson C., & Verno A. (2003). A model curriculum for k–12 computer science: final report of the ACM K–12 task force curriculum committee. https://dl.acm.org/doi/book/10.1145/2593247.
Tyler, R. W. (1949). *Basic principles of curriculum and instruction*. Chicago, IL: University of Chicago press.
Unahalekhaka, A., & Bers, M. U. (2022). Clustering young children's coding project scores with machine learning. In *Proceedings of the IEEE global engineering education conference (EDUCON)* (pp. 79–85).
Vee, A. (2017). *Coding literacy: How computer programming is changing writing*. Cambridge, MA: The MIT Press.
Walker, D. (2003). *Fundamentals of curriculum: Passion and professionalism* (2nd ed.). Mahwah, NJ: Lawrence Erlbaum.
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, 6, 20-23.