Check for updates

Review of Educational Research
Month 202X, Vol. XX, No. X, pp. 1–38
DOI: 10.3102/00346543241241327
Article reuse guidelines: sagepub.com/journals-permissions
© 2024 AERA. https://journals.sagepub.com/home/rer

Coding and Computational Thinking Across the Curriculum: A Review of Educational Outcomes

Kathy A. Mills
Jen Cope
Laura Scholes
Luke Rowe
Australian Catholic University

Teaching coding and computational thinking is an emerging educational imperative, now embedded in compulsory curriculum in the United States, Finland, the UK, Germany, Belgium, the Netherlands, New Zealand, and Australia. This meta-synthesis of 49 studies critically reviews recent international research (2009–2022) of coding and computational thinking as core and integrated across the curriculum. It addresses four essential problems: (a) What are the key features of learning environments that successfully develop students' coding and computational thinking? (b) What is the impact of student engagement in coding and computational thinking on learning outcomes across curriculum areas? (c) What pedagogical constraints are evident for coding and computational thinking, including across curriculum areas? and (d) Which conceptual frameworks support coding and computational thinking, and what has been marginalized or excluded? The review advances knowledge of coding and computational thinking—vital to guide and develop future AI-based solutions to real-world problems that challenge disciplinary boundaries.

Keywords: coding, computational thinking, game design, integrated curriculum, programming

Students across the world are growing up unprepared for rapidly changing digital environments where occupations that require the application of coding capabilities and computational thinking skills extend well beyond specialized technical industries (Shute et al., 2017). Technological developments with transformative potentials for education have burgeoned in the digital age, with initiatives targeting student competencies in computer coding or programming literacies and computational thinking—skills that have lifelong validity

(Vee, 2017). However, despite widespread agreement that coding and computational thinking skills at various levels of abstraction are increasingly relevant to multiple curriculum areas—such as mathematics, writing, and media literacy—coding still often assumes a minor, compartmentalized place in education. Additionally, there is limited research on its scope, potentials, and constraints for realizing educational transformation across knowledge domains. Systematic research is needed to inform and guide cumulative skill development for students across levels of education, from early childhood settings to elementary and high schooling, and to college or university (Lockwood & Mooney, 2017), with these skills increasingly important in jobs of the future to understand and develop AI (artificial intelligence)–based programs.

This meta-synthesis analyses research on coding or computer programming and associated computational thinking skills to identify pathways for education systems at all levels to design learning environments for coding that are underpinned by robust conceptual frameworks, to navigate the pedagogical constraints, and to guide the integration of computational skills to support learning outcomes across the curriculum in hybrid ways. Education systems need research to evaluate programs that work against the reduction of coding and its related thinking and problem-solving skills to niche technical skills to be taught through elective subjects—a position that is no longer tenable for workforce preparedness in the 21st century. Significantly, 52% of the current computer science and technologies workforce is employed outside technology-related industries, with a demand for technology skills increasing across many fields in a distributed knowledge economy (Deloitte Access Economics, 2017).

Historical Background

While the past decade has seen a rapid prioritizing of coding and computational thinking in the curriculum supported by educational clubs and after school programs, the earliest teaching of computer programming in schools began as early as the 1960s (Feurzeig et al., 2011). Logo, an early coding language, was developed by Papert and introduced to schools in the UK and United States in the 1980s, catalyzed by Papert's (1990) vision that Logo could be used as a tool to change the way we think about the relationship between mathematics and writing, engaging learners in processes of testing and debugging code. This approach was seen as a critical alternative to behaviorist approaches that were driving computer-aided instruction. However, like other educational reforms that are scaled up by bureaucratic industrialists, Logo was largely ineffectual in schooling, compounded by technical incompatibilities and the limited functionality of school computers of the time (Hickmott et al., 2018). Coding quickly receded as technology programs in education began to prioritize word processing in the 1980s and internet search skills in the 1990s (Popat & Starkey, 2019).

In the 21st century, coding and computational thinking became propelled forward in education by the work of theorists such as Wing (2006), and later others (e.g., Lye & Koh, 2014). These theorists argued for the urgent need for students to have facility with computational thinking: a kind of analytical thinking to solve problems, design systems, and understand human behavior, using concepts that are essential to computing (Wing, 2006). "Computational thinking" is defined

here as thought processes used to articulate solutions as computational steps that can be executed by a computer. Relatedly, "coding" refers to the act of programming or giving instructions to a computer (e.g., robots, chips, small devices), applying solutions developed through computational thinking. Subconcepts of computational thinking include thinking processes of pattern recognition, analysis, problem solving, abstraction, algorithmic thinking, decomposition, and automation; while core coding involves programming ability, such as using loops, variables, algorithms, and conditionals (Rich et al., 2021).

The emphasis on computational thinking was accompanied by a shift from playing games to students making digital games for learning—supported by developments in gaming trends, "do-it-yourself" (DIY) cultures, and a paradigmatic shift toward constructionist educational approaches (Kafai, 2018). Computational thinking and coding became centered in national curriculum agendas by 2016, when at least 18 countries had introduced coding and its related thinking strategies into the curriculum: the United Kingdom, Denmark, Austria, Malta, Poland, Spain, Slovakia, Portugal, Bulgaria, France, Israel, the Czech Republic, Estonia, Hungary, Lithuania, Ireland, Finland, and Belgium (Uzunboylu et al., 2017). Advanced computational thinking and problem-solving skills are associated with societal benefits such as increased social cohesion, reduced crime, and lowered welfare costs, impacting the health, well-being, and social mobility of future generations (Wolfe & Haveman, 2002).

Research Purpose

Given the global interest in computer coding, programming, game design, and competencies for computational thinking, a meta-synthesis of the collective contributions of research investigating coding conducted in educational settings, such as presented here, is much needed. The aim was to systematize the features of learning environments to develop coding expertise and other learning outcomes when integrated strategically across curriculum areas, whereby coding is embedded into subjects such as science, mathematics, history, or English to foster problem solving, critical thinking, creativity, or discipline-specific knowledge. To date, no published systematic reviews have examined the evidence, while original studies had pointed to this critical gap (Khanlari, 2016; Tofel-Grehl et al., 2021). Additionally, we sought to identify any pedagogical constraints to these outcomes, and to critique the underlying conceptual frameworks to guide the teaching of computer coding or programming, because the theoretical context always plays a role whether explicitly acknowledged or not (Alexander, 2020). The meta-synthesis systematically examines answers to the following four research questions (RQs):

RQ1: What are the key features of learning environments that successfully develop students' coding and computational thinking?
RQ2: What is the impact of student engagement in coding and computational thinking on learning outcomes across curriculum areas?
RQ3: What pedagogical constraints are evident for coding and computational thinking, including across curriculum areas?

RQ4: Which conceptual frameworks support coding and computational thinking, and what has been marginalized or excluded?

All the studies included in this review address both the first and second RQs, while 41 of the 49 studies address the third RQ on pedagogical constraints. Additionally, 41 of the 49 studies address the fourth RQ regarding learning approaches that underpin coding, game design, and associated computational thinking skills in the curriculum.

Method

The methodology of meta-synthesis was adopted—an interpretive analytical technique that uses the qualitative findings from a large set of peer-reviewed research to gain a broader and deeper understanding of phenomena than a singular original study. A meta-synthesis achieves this by searching for patterns across existing empirical studies (Sandelowski et al., 2006). The integration of qualitative findings in a meta-synthesis—typically from multiple countries, participant groups, and educational settings—generates new knowledge to answer original research questions through the synthesis, cross-comparison, and validation of empirical results (Shuval et al., 2011). Meta-synthesis was selected because of its ability to bring together multiple cross-national, contemporary studies that utilize a range of methodologies (Cooper et al., 2009).

Inclusion Criteria and Selection of Studies

Studies were included in this research meta-synthesis if they met the following eight criteria, which includes methodological requirements for selection: (a) identify key features of learning environments to develop students' coding and computational thinking—RQ1; (b) report on the impact of student engagement in coding and computational thinking on learning outcomes on at least one curriculum area—RQ2; (c) be published in English; (d) have undergone peer review; (e) be original research; (f) utilized qualitative or mixed methods approaches; (g) published between 2009 to 2022; and (h) focused on education from early childhood to post-secondary. Given previous concentration on P–12 populations (e.g., Kite et al., 2021; Li et al., 2020), extending the age range was an important inclusion criterion. To better understand the teaching of coding throughout the education system, studies were required to address the first two RQs in the context of education from early childhood through to postsecondary education. Further requirements for inclusion were that the research had to be original (primary sources), peer reviewed, and recently published (2009–2022). This time interval increases the comparability of the studies in terms of technologies for teaching coding, while maintaining a manageable data set against criteria of breadth, such as education levels from K–12 to postsecondary. It coincides with the increased availability of software such as Scratch and second-generation Lego Mindstorms to teach coding for younger learners. Consistent with the scope of a meta-synthesis and the descriptive nature of the RQs, we selected only qualitative or mixed methods studies. Quantitative studies are not included in a meta-synthesis, since they require statistical analyses of effect sizes, and the corpus of studies would be

too heterogeneous to be comparable. By excluding quantitative research, this review does not address comparative outcomes relating to specific dependent (e.g., standardized test outcomes) and independent variables (e.g., gender). Nevertheless, our screening of quantitative literature did not reveal any major themes that were omitted by this meta-synthesis. Examples of excluded studies include Martin (2017) because this source was not peer reviewed, as well as Thompson et al. (2018) and Choi (2015) because they were quantitative studies using multiple standardized instruments.

To identify studies of coding and educational outcomes, electronic searches were conducted using terms that included "computer," "coding," "game design," "maker space*," "robotics," "computational thinking," "computer literacy," and "computer programming." The Boolean operator "OR" was used where required to broaden the search using related terms, while "AND" was used to focus large returns to search for "coding" in specific fields (e.g., "education," "media," "animation," "writing," "math*," "science," "literac*," "multimodal"). Some search engines did not require the manual entry of Boolean operators.

Databases included ACM Digital Library, Academic Search Complete (EBSCOhost), Computers and Applied Sciences Complete (EBSCOHost), Education Source (EBSCOHost), Direct Open Access Journals, Elsevier Science, Emerald Education eJournal Collection, Learning and Technology Library Journals, Oxford Academic Journals, ProQuest Central, SAGE Journals, ScienceDirect, Scopus, SpringerLink Journals, Taylor & Francis Online, and Wiley Online Library—many of which include conference proceedings that were included in this review. Google Scholar was used to support the library database searches. Further, the citation pearl growing technique identified additional relevant research from bibliographies of included studies (Schlosser et al., 2006).

Data Extraction and Analysis

The team of researchers refined the data extraction table content and determined studies for inclusion, reading through full articles and proceedings to identify those that met the inclusion criteria, resulting in a total of 49 studies. We compiled a summary of each included study into a data extraction table, which was refined by the team with shared understanding of the categories. One researcher double coded the entries to ensure that the extraction tables provided sufficient detail on how the studies answered the RQs. The lead researcher then checked all entries. The table of data extractions for the 49 articles is available as Supplemental Table S1 in the online version only.

Tabulations were used to summarize and quantify the characteristics of the complete set of included studies using Excel spreadsheets. These categories included number of studies for each RQ; country of origin; curriculum areas; methodologies; participant ethnicities; socioeconomic backgrounds; age or level of education; education context (e.g., after-school club); and type of computer coding software used for programming, game design, or robotics. NVivo coding was then used to systematically code the central and recurring findings extracted from the complete data set for each of the four RQs, providing the basis for the results that are analyzed and discussed in this meta-synthesis.

Summary of Characteristics of Included Studies

The majority of studies were conducted in the United States (44.9%), with the country of origin not specified in 14.3% of studies. The next most frequent locations for the research were Spain (8.2%) and Canada (6.1%). Other countries included Israel, Italy, Spain, New Zealand, Greece, Russia, UK, Mexico, and Turkey. Some studies covered more than one of these nations. Table S2 in the online version of the journal presents a summary of the essential features of each of the 49 studies reviewed. The table of studies includes an article reference number, first author, publication year, country, participants, study focus, theory, methodology/methods, curriculum areas, teaching constraints, and strengths of the learning contexts and can be found at the following link: https://drive.google.com/file/d/1NdzX5viXNm0JJe9XhVF-Lw1QMIs7CkgE/view?usp=sharing.

The most commonly used methodology applied was case studies (48.9%), followed by interventions (22.4%), design-based research (10.2%), ethnographies (10.2%), exploratory or experimental (10.2%), and mixed methods (8.2%). More than half of the studies selected distinct methodological nomenclature not shared by the other studies, while others combined methodologies. The studies that met the criteria for inclusion spanned a wide range of curriculum areas with technology (34.6%) and STEM (science, technology, engineering, and mathematics; 34.7%) the most frequently occurring subjects for researching coding in education, followed by computer science (24.5%), mathematics (18.3%), English (14.3%), science (10.2%), and STEAM (includes arts; 8.1%). An additional third of the studies focused on other subjects, such as architecture, civic education, economics, e-textiles, history, humanities, media, music, and social studies.

With respect to education contexts, the largest group of studies was conducted in middle schools (28.6%), primary and elementary schooling constituted 24.5%, and secondary or high school cohorts made up 16.4% of the studies. Interestingly, 14.9% of the studies focused on teachers as the main participant group of interest; while a further 12.2% were studies that included diverse age ranges, for example, adults, families, or large age ranges (e.g., 7 to 18 or 14 to 25). University and college participants were the focus of 12.2% of the studies reviewed, contrasting with preschool or kindergarten participants in early childhood coding environments (4.1%). Note that these categories of education are not mutually exclusive, since some studies included multiple participant groups across different contexts. Across the studies that specified the numbers of research participants, a total of 2,542 students participated across the included studies, with a range of 1 to 324 participants per study. Three studies did not provide specific details of their participants but rather (a) counted numbers of student digital artefacts produced (n = 1,116; Literat & Kligler-Vilenchik, 2018), (b) focused on a single teaching unit (Grubbs, 2013), or (c) indicated only a range of participants across multiple sites (Sheridan et al., 2014).

In some studies, multicultural cohorts were the most frequently described participant group (10.2%), while the second largest group in the studies reviewed were African American or students of color (6.1%), followed by bilingual speakers of Spanish and English (4.1%). Other specific ethnicities included Indigenous

American (2%), Latina majority (2%), and other minority groups (2%). Other characteristics used to describe the participant groups, included students with particular learning needs or who were differently abled (10.2%), gifted students (4.1%), students from low socioeconomic backgrounds (8.1%), and students from diverse socioeconomic backgrounds (4.1%). Of the studies that identified the gender of the participants, the total number of identified males was 871 and females was 854, totaling 1,725. Studies with a male-female gender balance comprised 22.4%, while studies with a male majority constituted 38.8%. Contrastingly, studies with a female majority comprised only 12.2%.

Most studies did not indicate whether the participants had any previous coding or game design experience. Only one study specified that the research participants had diverse previous game design experience, while four studies involved only novice programmers with no prior experience. Several studies indicated that participants had some previous experience in interactive media design or system design. The learning experiences described in the studies spanned a wide variety of in-school and out-of-school contexts, with 100% of studies specifying the learning context, and seven studies including more than one research location. The largest group of studies focused on face-to-face classes and workshops (38.8%); a further 16.3% were conducted within online communities, such as through Scratch.

The second most frequently occurring learning context was after-school clubs and workshops, including elective classes conducted in conjunction with schools (12.2%). A smaller group of the studies was conducted in university or college-based lab programs (6.1%) attended by college-level students, while a further 6.1% were university-based lab programs attended by school children. Other contexts included summer school programs (4.08%), makerspaces (4.08%), libraries (4.08%), a hospital-based program, a website-based coding program with an online survey, teacher workshops, and a public event. The most frequently selected coding software used in the studies was Scratch (32.6%), followed by three equally used software systems: Lego (Mindstorms, Education WeDo, 6.1%), Game Maker Studio (6.1%), and Makey Makey (6.1%). Arduino LilyPad kits were also used in multiple studies (4.08%), while an additional 20 studies each used a lesser-known program that was not common to any other study.

Results

The results are presented in four sections, with each section synthesizing the findings of the 49 studies to address RQs 1 through 4.

Key Features of Learning Environments for Coding and Computational Thinking

The first RQ seeks to identify the central elements of successful learning environments to develop students' coding and computational thinking. The results are divided into subsections that attend to the prominent and recurring themes in the studies, that: (a) distributed expertise is essential; (b) student-driven learning works effectively with the support of timely explicit instruction; (c) problem-solving skills should be taught early; (d) predictive thinking should be balanced

with taking risks; (e) use coding and computational thinking to solve cross-disciplinary problems, and (f) utilize supportive online coding communities.

Distributed Expertise Is Essential

The need for distributed expertise in coding and computational thinking programs emerged as the most frequent finding among the features of successful learning environments for coding and computational thinking. *Distributed expertise* refers to cognition and knowledge that is distributed across a learning network and utilized, for example, when groups work together to provide feedback and support to one another (Bass et al., 2016). Students become expert novices in key domains of knowledge and tutor other students in their area of knowledge application. Learning experiences may be engineered to capitalize on this knowledge or arise serendipitously (Brown et al., 1993). Learning to code was consistently described as most productive when students were positioned as coconstructors of knowledge, with examples that involved community-centered digital making (Calabrese Barton & Tan, 2018). In contexts where students required learning support, imitation of more capable peer experts was effective (Bargagna et al., 2019). Student pairings were structured to optimize constructive and supportive task-aligned dialogue (DeVane et al., 2016).

It is important to note that the degree and nature of peer collaboration varied across the studies. While some interventions paired or grouped students to work together from the outset, in other contexts students coded individual projects, but involved peers by providing evaluative feedback, acting on feedback, acknowledging effort, asking for guidance, playtesting, remixing coding, or sharing their designs with peer audiences (An, 2016; Baytak & Land, 2010; Burke, 2012). Distributed expertise simultaneously involved both peers and adult experts—teachers, technical experts, community, and researchers—so that expertise was intergenerational (Price & Price-Mohr, 2018; Tzou et al., 2019). For example, in working on long-term independent digital media projects, underrepresented minority young adults were supported by peers and instructors with a unique combination of external industry experience and awareness of youth development (Bass et al., 2016). Instructors encouraged creative expression, personal agency, and sustained and supportive relationships through long-term community projects. Other examples include an after-school computational problem-solving lab with middle school students using Kodu Game Lab, where learners were partnered with supportive peers, while adult mentors provided high-level cognitive support to groups of struggling learners (DeVane et al., 2016).

A dominant theme was that within these intergenerational spaces emergent maker cultures were actively shaped in community, which worked across a varied array of maker projects and cultural contexts. For example, there were long-term coding club settings (e.g., four years) in which comaking was principally shaped by middle-school youth, utilizing their diverse interests and historicized cultural practices of communities as legitimate resources for making (Calabrese Barton & Tan, 2018). Likewise, in research with 146 economically marginalized high school teenagers (in which 93% were students of color), long-term student-driven 3D animation and game-making coding projects were aimed to improve cognitive elements, but also encouraged civic engagement

and personal voice. Students reflected on their identities; their communities; and their personal, political, and social worlds (Bass et al., 2016).

The same principles were applied in a programming workshop for high school students involving an online design challenge by Scratch at MIT to code animated music videos within a supportive learning community of mentors and peer-expert groups (Fields et al., 2015). In another intergenerational setting, Indigenous families were taught coding and robotics for restorative cultural storytelling, where intergenerational support proved successful when aligned with family and community interests. The outcome was that families stretched their skills far beyond basic coding because they were motivated to simulate key elements of cultural significance in their designs (Tzou et al., 2019).

Across the studies, lessons involving distributed expertise required strategic pedagogical work of teachers and other adults, particularly when introducing new concepts or technology devices. Teachers established and reinforced interaction rules, structures, and timelines to support peer collaboration, which was particularly important to sustain group participation in coding and computational thinking projects (Bargagna et al., 2019; Bass et al., 2016).

Student-Driven Learning Works Effectively With the Support of Timely Explicit Instruction

A consistent finding was that learners made significant gains in programming concepts when student agency allowed for the student-driven creation of coding projects in which student ownership and student choice were high, but where timely explicit instruction was also provided. This included contexts where students were self-directed and actively engaged on their own terms, often choosing the focus of their programming projects to reflect their interests. In such contexts, students developed an ability to code events; apply parallelism, logic, loops, and computational practices; while maintaining motivation and commitment to coding and computational thinking challenges (Sáez-López et al., 2016).

Timely explicit instruction by teachers and other mentors was needed to advance students' high-level coding, such as to use cleaner and more efficient logic and coding strategies (Denner et al., 2012). This approach involved systematically demonstrating to students the key ideas and steps required to learn something new. Teachers broke down complex tasks into simpler steps, used worked examples, provided opportunities for students to practice with explicit guidance, and frequently clarified learning objectives (Stockard et al., 2018). Using this approach, fifth graders developed stronger reasoning skills using Game Maker, which blended student ownership of learning with strategic scaffolding and guidance by teachers (Baytak & Land, 2010). Similarly, junior high robotics students using Lego Mindstorms appreciated short, focused segments of teacher instruction to support their student-driven projects because they were able to immediately put new knowledge to use, saving them considerable time (Barak & Zadok, 2009). Effective coding interventions used explicit instruction to analyze, annotate, and discuss code (Ascenzi-Moreno et al., 2020), while students with no prior coding experience required more extensive instructional support, particularly for understanding high-level computational thinking concepts, more efficient coding, and debugging (Denner et al., 2012).

The optimal ratio between student-driven coding and teachers' use of direct instruction was found in a study of second-grade students learning robotics. Teachers provided only short periods of instruction because students were impatient to progress with their projects. Teachers kept students on task, managed time, pointed students to resources, and helped those who had missed classes to catch up (Egbert et al., 2021). Other research with middle school students emphasized the importance of the teachers' explicit guidance, especially at the beginning of robotics problem-solving projects, to activate students' prior knowledge and to establish the task structure (Grubbs, 2013); and while students exercised ownership of projects, adequate teacher expertise was vital (Khanlari, 2016). Approaches were primarily student-driven—emphasizing student agency and self-direction—supported by the teachers' mini-instructional lessons, incidental teaching, valuable conversations, social support, and timely reminders. Rigid traditional models of teaching were found to impose limits and reduce student motivation, and too much teacher assistance notably reduced troubleshooting skill development (Hughes & Morrison, 2018; Kafai et al., 2010; Lindsay & Hounsell, 2017).

Teach Problem-Solving Skills Early

A key finding was that problem-solving skills need to be taught early in schooling—not introduced for the first time in secondary schooling, a postsecondary career, or technical training. For example, research with low-income, underrepresented participants found that students were not exposed to problem-based instruction early enough to develop their capacity sufficiently over time (Bass et al., 2016). In contrast, coding success was demonstrated in research in early childhood classrooms involving the use of Kibo robotics kits, which showed that very young learners could grasp foundational skills involving coding in sequences, using loops and conditionals, and debugging (Bers et al., 2019). Likewise, groups of students aged 5 and 6 in a kindergarten classroom using Logo-based computer programming of games were able to propose alternative, shorter coding solutions to those of previous players (Fessakis et al., 2013).

Predictive Thinking Should Be Balanced With Taking Risks

Another important finding was that predictive thinking in coding and problem-solving needs to be balanced with encouragement to take risks with programming events. Coding sequences are then modified based on an evaluation of the playtesting or action. Playtesting was often performed by peers as an important stage of the design process across the studies. While certain apps, like Scratch Junior, can encourage experimentation in coding and computational thinking methods, operationalizing the teaching of problem solving requires a supportive learning culture where students are encouraged to ask for help and can make errors (Falloon, 2016). In a junior high college robotics class using Lego Mindstorms, many students were able to solve problems intuitively but needed help to articulate and reflect on the problem-solving process rather than relying on lengthy cycles of trial and error (Barak & Zadok, 2009).

Use Coding and Computational Thinking to Solve Cross-Disciplinary Problems

Importantly, many studies demonstrated the effective teaching of coding and computational thinking to solve cross-disciplinary problems. For example, a

study of teaching physics through coding and e-textiles to high school students in Grades 9 through 12 provided opportunities for students to debug and troubleshoot programs. The richly complex intersection of art, design, coding, and mathematics led students to engage across multiple disciplinary areas to solve problems (Tofel-Grehl et al., 2021). Similarly, science-technology research with secondary students (13–15 years) in a science center workshop made augmented board games with Scratch (Kafai & Vasudevan, 2015). Pairs completed debug'ems—Scratch programs with intentional faults—which were directly connected to the functionality of their digitally supported board games. The students learned to use sequences, events, operators, parallelism, conditionals, and loops, connecting problem solving with knowledge beyond the technology curriculum.

Utilize Supportive Online Coding Communities

A key finding was that students learning to code prefer to have a genuine online audience for their programs to enhance motivation, confidence, and pride in their coding and computational accomplishments. For example, for young interactive media designers (ages 7–18), the desire for an audience was important for sharing their work online. Students engaged in valuable online conversations that informed their identities and pride as programmers and received acknowledgement of their efforts from the wider coding community (Brennan, 2016). Other studies found that students were disappointed when their shared projects did not attract attention, but were excited when online comments were received, especially those that originated beyond the school (Burke, 2012).

Interestingly, the confidence and skills to navigate online portals, such as the online Scratch community, need to be nurtured early in students' coding and computational thinking development. For example, Year 6 students who learned to code using Scratch in an after-school club needed a certain level of comfort and familiarity with the site before being confident to share their coding. Students were also cautious about remixing code from online participants, preferring to observe rather than reuse others' code. Students' willingness and degree of engagement with others in online coding communities varied, with some who friended other users, commented on projects, and uploaded their own projects; while others did not seek to share projects or seek feedback (Kafai et al., 2010).

Supportive online communities fostered communication beyond the immediate classroom to receive feedback from wider online "affinity groups" with shared coding interests (Gee, 2003). An example is a multisited study of an after-school programming Collab Challenge, where high school students (aged 14–18 years) coded games, animations, and stories that they shared with members of the broader Scratch community. Participants were motivated to engage with online communities to share their work, to receive recognition, and to take responsibility for helping others (Kafai et al., 2012). This and other research (e.g., Hagge, 2017) highlighted the value of online coding communities for students' self-expression, where participants gave and received affirmations and encouragement. However, there were challenges in online spaces, such as handling negative comments and unrelated political posts, and students choosing to later remove posted content. This was the case with research by Literat and Kligler-Vilenchik, (2018), who worked with Scratch

members aged 8 to 13 years of age. Students reacted negatively to some online content from peers, with younger students notably less equipped to critically evaluate online comments.

Other studies reported ambivalence among students about online sharing of coding, contributing to mixed results in terms of the nature and level of support provided by online communities (Fields et al., 2015; Vasudevan et al., 2015). Some participants perceived the degree of online support as relatively minor compared to face-to-face support. While the online communication was less formative in shaping students' work than regular peers and mentors, students felt that the online community was generally encouraging, while some modified their projects based on constructive online feedback (Fields et al., 2015).

In terms of the opportunities that online communities such as Scratch provide for remixing code, middle school youth (aged 11–13 years) engaged in this practice to varying degrees, from simple copying to more creative extensions of code, performed at diverse levels of competency. Overall, remixing online code created by online users was found to enhance coding and computational thinking at school, particularly for beginning programmers (Fields et al., 2015; Hagge, 2017; Literat & Kligler-Vilenchik, 2018).

Impact of Coding and Computational Thinking on Learning Outcomes Across Curricula

Coding literacy for students was a key driver across the studies, but in answering our second RQ, we found there was an emphasis on four learning outcomes identified as (a) computational thinking for problem solving, and creative and critical thinking; (b) disciplinary knowledge through coding; (c) student agency, self-direction, motivation, and active engagement; and (d) social and collaborative skills and taking diverse perspectives. In the following four sections, we illustrate the findings for RQ2 by detailing the results within these subcategories.

Computational Thinking, Problem Solving, and Creative and Critical Thinking

Many of the articles positioned computational thinking for problem solving, along with creative and critical thinking, as key learning outcomes of the research. Of the 49 articles reviewed, while there was some integration of the desired learning outcomes, 9 studies identified computational thinking for problem solving as central (e.g., Baytak & Land, 2010; Kafai & Vasudevan, 2015), 21 studies were specifically coded for developing students' creative thinking or fostering creativity (e.g., Hughes & Morrison, 2018; Sheridan et al., 2014), and 7 studies identified critical thinking or evaluative skills (e.g., Fields et al., 2015; Ward et al., 2014).

While coding and computational thinking were often taught across the curriculum, most interventions were centrally focused on the successful development of students' higher-order thinking skills, critical thinking skills, heuristic strategies, predictive thinking, risk-taking and experimentation with programming, enhanced algorithmic thinking, development of more efficient or shorter code, debugging skills, and improved conceptualization of programming logic. Researchers observed enhanced collaboration, communication, questioning, confidence, articulation of knowledge, and interpersonal skills, coupled with the development of

personal attributes, such as perseverance or tenacity when encountering new challenges (Barak & Zadok, 2009; Bers et al., 2019; DeVane et al., 2016; Falloon, 2016; Fessakis et al., 2013; Grubbs, 2013; Kafai & Vasudevan, 2015; Tofel-Grehl et al., 2021).

Coding was deemed a vehicle for fostering problem solving in a range of fields, such as mathematics, technology, or engineering, as students used critical thinking based on real-life experiences and interdisciplinary problems (Barak & Zadok, 2009; Egbert et al., 2021; Sacristán & Pretelín-Ricárdez, 2017). Developing problem solving through robotics, for instance, highlighted how technology education is often embedded across disciplines through STEM curricula. Robotics emerged as the most frequently researched activity for teaching problem solving, with programming and playtesting code an essential feature of these environments (e.g., Barak & Zadok, 2009; Bass et al., 2016; Bers et al., 2019; DeVane et al., 2016; Grubbs, 2013). For example, a study used robotics to facilitate learning related to engineering design processes as students solved authentic problems to think critically to plan, design, and evaluate a robotics program for a local council to use to clean a sports stadium (Grubbs, 2013). Similarly, students were engaged in thinking critically to elicit solutions to problems in a study that engaged second-grade students to design codes for robot pathways (e.g., turning, going faster), while thinking creatively to design their own tracks for the robots (Egbert et al., 2021). Developing knowledge of basic robotics was also used to improve polytechnic orientation of the academic process, where robotics was considered a special teaching technology (Ospennikova et al., 2015).

Other contexts involved coding and debugging with Scratch and Scratch Junior (e.g., Falloon, 2016; Kafai & Vasudevan, 2015), other digital game design programs (DeVane et al., 2016; Fessakis et al., 2013), and making e-textiles (Tofel-Grehl et al., 2021). Computational participation involved projects that fostered creative skills to design and remix, alongside traditional conceptions of coding and programming through digital game creation with tools, such as Kodu and Scratch, to engage with abstract problem solving (DeVane et al., 2016).

Fostering problem solving, critical thinking, and creativity was an important goal across the studies. For example, coding was embedded in the social studies curriculum when 7th-grade classes created different sprites to develop historically accurate games on containing communism. Aimed to be challenging and fun, coding equipped students to think critically and creatively (An, 2016), skillsets with cross-disciplinary application (Abrami et al., 2015).

Disciplinary Knowledge Through Coding

Advancing student disciplinary knowledge was clearly an important outcome of many of the coding and computational thinking studies. Knowledge goals within STEM were identified in 42 cases (e.g., Grubbs, 2013; Khanlari, 2016; Sacristán & Pretelín-Ricárdez, 2017; Tofel-Grehl et al., 2021). For example, Khalili et al. (2011) focused on developing key concepts in immunology with high school students through game development in STEM. Seven of the articles were double coded within the four disciplines; for instance, three studies were coded for both science and technology (Aragon et al., 2009; Baytak & Land, 2010; Pinto-Llorente et al., 2018). There were also cases where coding and related

problem-solving skills were used to develop disciplinary knowledge beyond STEM that were related to architecture, civic education, economics, e-textiles, history, humanities, media, music, and social studies—but were also cross-categorized for integrating elements of disciplinary knowledge related to technology (e.g., An, 2016; Gestwicki & McNely, 2012; Literat & Kligler-Vilenchik, 2018; Ratcliff & Anderson, 2011; Sáez-López et al., 2016).

Some studies examined the potential of coding to support language expression, providing an approach in language arts where code was used to compose digital stories, games, animations, and integrated literacy activities (Ascenzi-Moreno et al., 2020; Brennan, 2016; Parry et al., 2020; Price & Price-Mohr, 2018; Sáez-López et al., 2016). For example, a code-integrated language arts project used Scratch to facilitate middle school student storytelling, with the application of computational thinking practices, like abstraction and debugging, to build on the idea that code is not experienced in a vacuum (Ascenzi-Moreno et al., 2020). The idea that code can be annotated, discussed, and used in a story, connected to language, literacy, and other aspects of their lives and communities, was a pedagogical approach used to develop both language arts and computing skills for digital media skills (Brennan, 2016; Hagge 2017) and multimodal authoring (Ascenzi-Moreno et al., 2020; Parry et al., 2020; Price & Price-Mohr, 2018; Sáez-López et al., 2016).

In another instance, Gestwicki and McNely (2012) worked with university undergraduates using a five-phase design thinking model as a lens for producing a serious game for an educational museum. The cycle consisted of empathy, problem definition, ideation, prototyping, and testing and was embedded in a program to create a game with Root Beer Float Studio in Muncie, Indiana, in cooperation with The Children's Museum of Indianapolis. The game included the player in the role of a museum volunteer who was tasked with creating digital exhibits from the museum's collection with the aim to develop coding, designing exhibits, and thinking for game-based designs.

In our review, we identified 13 studies specifically focused on teaching cross-disciplinary outcomes, generating evidence of the value of coding for learning across the curriculum. For example, one study that focused on language arts–based coding taught multimodal authoring skills and critical media literacies (Price & Price-Mohr, 2018). The affordances for STEM-arts practices, grounded in storytelling, engaged cultural ways of knowing while cultivating learning outcomes in STEM through the design of new technologies (Tzou et al., 2019). As interdisciplinary learning makes for astute problem solving, these studies offer important insights that lay the groundwork for future AI-based coding and other forms of mathematical thinking taught with computational thinking, such as probability, statistics, and measurement (Hickmott et al., 2018).

Student Agency, Self-Direction, Motivation, and Active Engagement

Coding and computational thinking experiences across the studies demonstrated opportunities to exercise agency, self-direction, motivation, and to be actively engaged in the learning process. Activities encouraged creative expression, personal agency, and pursuing interests (Bass et al., 2016), with attention to how pedagogies for coding can enhance a strong sense of responsibility and

autonomy (Franetovic, 2016). Student agency was facilitated through trial-and-error strategies that involved iterations of debugging. Students applied conceptual models about how to use space, resources, and time, with students becoming highly motivated and engaged (Ratcliff & Anderson, 2011; Sacristán & Pretelín-Ricárdez, 2017; Sáez-López et al., 2016; Tonbuloğlu & Tonbuloğlu, 2019).

Significant improvements in learning programming concepts, logic, and computational practices were noted in a study of fifth- and sixth-grade students. Sáez-López et al. (2016) used Scratch in classrooms with 107 students. The pedagogical design involved students in creating their own stimulating content in sciences and arts. The intrinsic sense of reward, and sense of ownership of learning overcame issues related to disability and gender, was demonstrated in a study involving problem solving with fourth graders (Ratcliff & Anderson, 2011). This also highlighted the value of early experiences (Buitrago Flórez et al., 2017).

A focus on media production also offered opportunities for self-direction and engagement. Parry et al. (2020) show that engaging students in media production in which participants create video games/stories can also enhance student autonomy. Identifying pedagogical approaches to support this, McDougall and Potter's (2019) application of the "third space" (Gutiérrez, 2008) provides a useful framework for understanding opportunities for productive disruptions as contested, negotiated, and political. Young people selected their own ideas for stories, developing self-direction and a sense of agency, particularly as they had the option to write themselves into fictional texts. As society transitions to Industry 5.0—where humans and machines combine—fostering student agency, self-direction, motivation, and active engagement will be vital to produce unique and customized outputs.

Social and Collaborative Skills and Taking Diverse Perspectives

Coding and computational thinking offered scope to advance participatory cultures, facilitating opportunities to promote interest-driven communities where students come together to create, remix, and share their work (Grimes & Fields, 2015). Programs such as Scratch were used to provide dynamic spaces for young programmers to create, comment on peers' work, and seek help in supportive forums and sharing expertise to enhance the product (Fields et al., 2015). Such programs support collaborative environments for unique coding, debugging to resolve challenges, and engagement with computational concepts as a team (Kafai et al., 2012; Kafai & Vasudevan, 2015). Participation in online learning communities was also found to facilitate the development of interpersonal skills and communication (Hagge, 2017).

Students learned the value of taking diverse perspectives when designing a game or playtesting, and from iterations of feedback (An, 2016; Sheridan et al., 2014; Tzou et al., 2019). Shifts in classroom dynamics from teacher-centered or initiated to learner-centered—with a focus on distributed knowledge—can create more space for students to feel a sense of efficacy and their social presence in the classroom and learning process, building a community of practice (Hughes & Morrison, 2018). Game-making allowed students to enhance their problem solving as they conceptualized the integration of game elements, realized computational concepts and practices in their game designs, and reflected on their game design experience.

Social and collaborative skills were fostered through shared interests (Kafai & Vasudevan, 2015), the need for teams or communities to achieve goals (Brennan, 2016; Tonbuloğlu & Tonbuloğlu, 2019), and through students' combined efforts to create (Grimes & Fields, 2015). Bass et al. (2016) showed student engagement in several phases of game development involving distributed expertise from peers and their instructor—building a community of practice through peer feedback. Other studies embedded coding within literacy (Parry et al., 2020), science (Tofel-Grehl et al., 2021), and the arts (Sáez-López et al., 2016), pointing to the social aspects and opportunities to foster alternative points of view.

Pedagogical Constraints for Teaching Coding and Computational Thinking

Constraints for teaching coding and computational thinking were identified in most of the original studies (84%, 41 out of 49), with just eight of the studies not addressing this question (RQ3). Most studies (80%) raised multiple concerns encountered when teaching coding. Some studies drew attention to up to 10 different barriers and constraints occurring in these education innovations (see Egbert et al., 2021; Gaeta et al., 2019; Khanlari, 2016; Tonbuloğlu & Tonbuloğlu, 2019). Across the 41studies, a total of 131 different teaching and learning constraints that negatively impacted the teaching of coding and computational thinking were identified.

This section presents the results for the third RQ—to identify and discuss the pedagogical constraints for teaching coding and computational thinking. It reviews the main types of constraint recurring across the original studies, from most to least common: (a) lack of teacher and student skills, knowledge, experience, and creativity, (b) technology context difficulties, (c) social communication problems, (d) barriers to developing student and teacher coding proficiency, (e) general learning environment problems, and (f) time constraints.

Teacher and Student Lack of Skills, Knowledge, Experience, and Creativity

Coding constraints associated with limited coding knowledge, slow improvement, or limited creativity constituted the most frequent type of problem and applied both to teachers and students. The range of factors has significant implications for teaching coding, from the dual perspectives of teachers and students. For teachers, the most common constraint concerned their initially low valuing of coding in projects (17.9%), which was mitigated to some extent by researchers working with a project teacher (Egbert et al., 2021; Khanlari, 2016).

Limited support for teacher practice and professional development or training was also frequently noted (see, for example, Bers et al., 2019; Israel et al., 2015; Lindsay & Hounsell, 2017; Pinto-Llorente et al., 2018). A smaller number of teachers discussed how their limited experience with coding impacted their confidence to learn and teach these skills (e.g., Rich et al., 2021). Some teachers experienced stress (Egbert et al., 2021), could provide only limited teacher role modelling (Denner et al., 2012), and lacked coding-related computational thinking proficiency (Snodgrass et al., 2016).

Teachers were critical of some of the primary students' coded stories in a study that looked at manipulating code to represent story meaning. The teachers were unimpressed with emerging stories in which story characters were

underdeveloped and stories lacked appropriate grammar and words to create interest (Price & Price-Mohr, 2018). In another study that used robotics for educational purposes, elementary school teachers perceived that the program did not prepare students for mandated outcomes and tests; the topic was unnecessary; and the subject was costly with respect to the time, resources, and effort required (Khanlari, 2016).

Student-related learning constraints were identified. For example, a study by An (2016) illustrated how middle school students who were designing and creating their own games were surprised to discover the amount of creativity and imagination the task required. Teachers identified that a group of mostly novice programmer middle school students were unwilling to use some computer science concepts, which necessitated greater instructional support (An, 2016; Denner et al., 2012). A further example was students' inability to set up contexts and objectives of the games: They needed to program with greater empathy for the experiences of the video game player. This was identified in a study in which university students designed an educational video game on how museums operate (Gestwicki & McNely, 2012).

Students' learning of coding was constrained by their lack of content knowledge (Fessakis et al., 2013; Khalili et al., 2011) or the difficulty in integrating other subject knowledge—such as mathematics—into coding (Tonbuloğlu & Tonbuloğlu, 2019). Similarly, a lack of task structure frustrated efforts to simultaneously develop computation techniques and problem-solving abilities (e.g., DeVane et al., 2016; Ratcliff & Anderson, 2011). Deficiencies found across the studies were teachers' low valuing of coding, limited teacher training, or students' lack of content knowledge or insufficient task support—each detrimental to the development of coding and computational thinking abilities (Bers et al., 2019; Denner et al., 2012; Egbert et al., 2021; Fessakis et al., 2013; Khalili et al., 2011; Khanlari, 2016; Pinto-Llorente et al., 2018; Rich et al., 2021).

Technology Context Constraints

A considerable proportion of studies specified technological factors as an impediment to teaching and learning coding and computational thinking, with coding tasks requiring technical expertise to overcome problems with computer hardware, software, or the internet that was beyond the reach of teachers (Baytak & Land, 2010; Egbert et al., 2021; Khanlari, 2016). This was coupled with students performing inefficient coding actions, such as continual reframing and testing (DeVane et al., 2016), or students making random adjustments to code (Falloon, 2016).

General technical problems included poor Wi-Fi connections and log in problems, and network support that was limiting for multiple devices (Gaeta et al., 2019). Students cited difficulties with the process of publishing games online, finding that progress required a considerable amount of patience (Frydenberg, 2015). Some platforms supported limited actions, potentially contributing to unimpressive coded stories (Price & Price-Mohr, 2018).

A further cause for concern was the difficulty inherent in some coding software, which resulted in students struggling to use critical coding elements or to meaningfully combine them (Theodosiou & Karasavvidis, 2015). Certain

programming issues caused students to give up in some cases (Pinto-Llorente et al., 2018). Other studies noted that coding technology difficulties discouraged students from exploring the remixing process (Kafai et al., 2010), or from debugging (Kafai et al., 2014). Some students chose not to make their video interactive (Fields et al., 2015), while others overengaged with technology (Egbert et al., 2021). The technology's novelty for students was also found to diminish by the end of two program cycles (Gaeta et al., 2019). These examples illustrate how technological limitations, including inherent difficulties of coding infrastructure and software, can create issues that frustrate student progress to develop their coding abilities (Baytak & Land, 2010; Egbert et al., 2021; Frydenberg, 2015; Gaeta et al., 2019; Kafai et al., 2010, 2014).

Social Communication Problems

Several constraints related to social communication problems, especially within online communities. There were general social communication difficulties online and offline (e.g., Bargagna et al., 2019; Sheridan et al., 2014), challenges specific to the online activity (Franetovic, 2016; Kafai et al., 2010), and social difficulties—such as shared online projects being ignored (Brennan, 2016), competitiveness among friends (Fessakis et al., 2013), and disagreement among teams (Pinto-Llorente et al., 2018). In other studies of coding offline, student interactions in group work were identified as a constraint when highly engaged students became intolerant towards their less motivated peers (Franetovic, 2016).

Online community-derived constraints were identified on the Scratch programming platform, with students experiencing disappointment and confusion when their online efforts were ignored, and students avoiding potentially negative attention from the Scratch community. This discouraged students from sharing their work, while prompting others to remove posts (Brennan, 2016; Literat & Kligler-Vilenchik, 2018). Working with online communities was also found to effect a change in the nature of the project coded by participants. For example, students changed their personal stories to video games to safeguard the online exposure of their work, while their personal stories did not transfer well to the Scratch platform (Burke, 2012). Combative comments or misinformation posted online resulted in participants being reported or blocked for causing offence (Literat & Kligler-Vilenchik, 2018). Approximately 40.8% of identified constraints across studies addressed unforeseen issues for students communicating online, pointing to the need to carefully monitor and support online interaction (Brennan, 2016; Burke, 2012; Franetovic, 2016; Kafai et al., 2010; Literat & Kligler-Vilenchik, 2018).

Barriers to Student and Teacher Learning Progress in Coding and Computational Thinking

Cognitive-based coding difficulties, for both teachers and students, was an even more challenging obstacle than the deficiency-type constraint reviewed above ("Teacher and Student Lack of Skills, Knowledge, Experience, and Creativity").

The most cited learning struggle (15.8%) related to the complexity of coding tasks (Bargagna et al., 2019) or complex programming activities, which involved

concepts beyond the understanding of young novice programmers. These were tasks such as incorporating global variables to earn points in a computer game (Denner et al., 2012), and difficulty processing instructions, which contributed to the participants' less proficient take-up of more complex coding and computational thinking (Bargagna et al., 2019; Frydenberg, 2015).

The next most cited learning ability constraint centered on students' inability to transfer skills (or concepts) from one context to another. For example, problem solving is often domain-specific, making it difficult for students to transfer such learning across contexts (Barak & Zadok, 2009; Falloon, 2016). Several individually identified constraints related to the differing amount of perceived student support, for example, student struggles due to teachers' use of computational thinking pedagogy, or specific types of learning support for students needing significant help (Frydenberg, 2015, p. 260; Snodgrass et al., 2016).

Some research studies found that a poor enactment of discovery learning led to chaotic procedures, such as in the case where students using Logo coded randomly and impulsively (Ratcliff & Anderson, 2011). One study found that freedom of choice can paralyze some students with learning disabilities and, conversely, too much assistance deters students from experiencing failure and taking troubleshooting risks (Hughes & Morrison, 2018). Other constraints were students' infrequent use of sophisticated programming (Fields et al., 2015), unequal participation in group work (Kafai & Vasudevan, 2015), reduced motivation (Hughes & Morrison, 2018), and unfinished games (Khalili et al., 2011).

Students became unmotivated when they could not see their own coding and computational thinking progress or if learning progress was slow (Tonbuloğlu & Tonbuloğlu, 2019). Teachers also had trouble incorporating design concepts into their own game designs, producing low-quality products that failed to meet the minimum assessment criteria (Theodosiou & Karasavvidis, 2015). Cognitive-based difficulties—such as student inability to understand and integrate complex coding tasks—together with issues of teacher support were key issues impacting students' coding and computational thinking development (Denner et al., 2012; Falloon, 2016; Frydenberg, 2015; Hughes & Morrison, 2018; Ratcliff & Anderson, 2011).

General Learning Environment Problems Arising in Coding Lessons

Problems that can arise in teaching or learning environments resulting in low student engagement and off-task behavior occurred in similar proportions to cognitive-based difficulties (see previous subsection). For example, negative behaviors observed in coding classrooms included low student engagement (Franetovic, 2016) and student distraction (Egbert et al., 2021). Overcrowded classes were occasionally problematic vis-à-vis scheduling activities and teamwork (e.g., Tonbuloğlu & Tonbuloğlu, 2019). Further concerns related to aspects of the classroom or workshop environment itself. For example, in one study, the sterile computer room situated away from the public library space limited the youths from negotiating the disruptions they needed to make their video games. This was additional to the youths' recognition that their video games' sounds were incompatible with the library's requisite silence (Parry et al., 2020). Another study, which explored best practices among young adolescent learners

using e-textiles, highlighted that a "stricter" teaching style is problematic. The researchers found that rigid approaches constrained the students' motivation and engagement by imposing purpose—such as having to create a particular project—instead of offering a broader range of projects (Hughes & Morrison, 2018). They recommended that once students have understood the theoretical concepts of makerspace tools, they should choose their own end product to generate internal motivation.

Issues related to underuse or overuse of learning materials and resources to teach coding were also identified. In one study, well-written instructional materials were available, but failed to enable middle school students to understand or incorporate coding concepts, such as variables, into their computer games (Denner et al., 2012). In some cases, programming documentation, mobile app templates, and standardized methods of presenting information were lacking (Denner et al., 2012; Ward et al., 2014), while other learning materials were excessively detailed (Tonbuloğlu & Tonbuloğlu, 2019). In contrast to sole reliance on coding materials (Rich et al., 2021), well-designed professional development programs heightened teacher coding confidence, with consequent successful teacher support for students coding their projects.

Overall, problems observed in coding lessons or workshops—similar to those arising in other curriculum areas—included learning challenges, class size and ambience, timing of lessons, teaching styles and student motivation, and underreliance or overreliance on instructional materials (Bers et al., 2019; Denner et al., 2012; Egbert et al., 2021; Franetovic, 2016; Hughes & Morrison, 2018; Rich et al., 2021; Tonbuloğlu & Tonbuloğlu, 2019; Ward et al., 2014).

Time Constraints

Finally, time-related constraints presented a barrier to teaching and developing effective coding skills, notably in school and university programs. One third pointed to limited or inadequate availability of time in the curriculum. For example, a 10-week program for fifth graders to examine the effect of unplugged coding activities found that the allocated time was insufficient for students to implement their activities and their problem-solving skills did not improve (Tonbuloğlu & Tonbuloğlu, 2019). Time was problematic in Grade 2 classrooms, where coding lessons took place using Ozobot and Ozoblockly, which required a long time to set up and organize materials to teach coding (Egbert et al., 2021). Time constraints impacted opportunities to demonstrate content-area knowledge, such as fifth graders designing computer games to understand nutrition, who had insufficient time to incorporate all functions in their pregame design (Baytak & Land, 2010). A study on the effect of continuous professional development (5 day-long sessions over 1 year) on coding in elementary schools found that more structured time was required to teach coding effectively (Rich et al., 2021). With time constraints documented in 35% of the studies, a greater amount of time needs to be built into professional development for teaching coding and computational thinking across the curriculum.

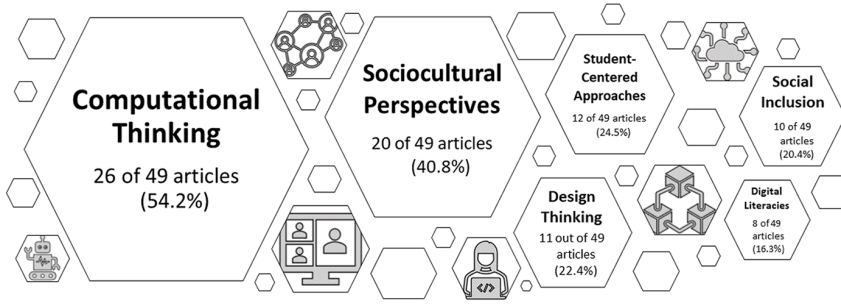

FIGURE 1. *Six conceptual frameworks for coding and computational thinking.*

Conceptual Frameworks Used to Support Coding in Education and Gaps in Knowledge

In this section, we examine the major conceptual frameworks used to support inquiry into coding in various educational contexts. We draw inferences about which frameworks have been marginalized and identify emerging concepts that may point to future trends. Each conceptual framework was independently grouped by broader categories that were based on their theoretical, conceptual, and methodological links. For example, notions of "constructivism" and "social constructivism" were situated within the broader category of sociocultural frameworks. A similar process was used when overlapping concepts were implied by, for example, referencing theories by Papert (1990), Kafai (2018), or Piaget and Cook (1954) in relation to constructivist and sociocultural approaches; and Wing (2006), Brennan (2016), or Resnick (2012) in relation to computational thinking. Overall, our approach yielded six conceptual frameworks that we used to organize our inquiry into the relevant literature: concepts relating to computational thinking, sociocultural theory, student-centered approaches, design thinking, digital literacies, and social inclusion (see Figure 1, noting that hexagon size is representative of the proportion of articles).

Applying a Computational Thinking Lens to Understanding Coding

Of the 49 articles in our review, 26 (54.2%) drew upon computational thinking approaches to guide their research inquiry. Researchers such as Brennan (2016), Resnick (2012), Kafai (2018), Papert and Harel (1991), and Wing (2006) were frequently cited for having influenced the studies that applied a computational thinking approach to understanding coding. Brennan and Resnick's (2012) framework, for example, featured prominently among these studies, characterizing computational thinking as comprising "computational concepts—the concepts designers employ as they program, computational practices—the practices designers develop as they program, and computational perspectives—the perspectives designers form about the world around them and about themselves" (p. 3).

Among the most salient claims of those who used a computational thinking lens pertained to students' domain-specific knowledge and skills. While somewhat dependent on the type of coding environment used during learning (e.g., Scratch, Game Maker, LEGO Mindstorms), these benefits nevertheless shared similarities across studies. These included capabilities affecting the students' ability to code with structure, timing and sequencing, feedback loops, conditional logic, parallelisms, testing and debugging, and reusing and remixing (see Kafai & Vasudevan, 2015; Pinto-Llorente et al., 2018).

A stronger set of claims informed the proposition that employing a computational thinking approach can contribute broader benefits to higher-order thinking constructs, such as systems thinking, problem solving, critical and creative thinking, self-regulated learning, and applied algorithmic thinking—where students learn to break problems down into smaller solvable elements (e.g., Falloon, 2016; Israel et al., 2015; Khalili et al., 2011; Pinto-Llorente et al., 2018). These domain-general benefits and thinking skills strengthen motivation, satisfaction, and self-efficacy among students and teachers (Rich et al., 2021).

An explanation as to why adopting a computational thinking framework is so appealing is because it can cultivate skills that are transferable to future careers in computing (Denner et al., 2012). Others contend that computational thinking helps students master "universal academic actions" and complex cognitive processes (e.g., imagination, speech, memory), and form interdisciplinary links between fields such as mathematics, science, and the humanities (Ospennikova et al., 2015, pp. 24–25). However, Barak and Zadok (2009) dispute claims about generalizability, arguing that knowledge about coding is best advanced within the bounds of current and meaningful learning rather than for the purposes of distant topics or future careers.

Applying a Sociocultural Lens to Coding and Computational Thinking

Notions of computational thinking were often situated alongside conceptual frameworks arising from sociocultural theory, such as constructivist and social constructivist approaches. Of the 49 articles in our review, 20 were identified as having drawn upon sociocultural theory to guide their research inquiry (40.8%). Mercer and Howe (2012) argue that sociocultural theory enables us to see that "knowledge is not just an individual possession, but also the creation and shared property of members of communities who use 'cultural tools' (including spoken and written language), relationships, and institutions (such as schools) for that purpose" (p. 12).

Research that applied a sociocultural lens cited inspiration from authors of the early to mid-20th century, such as Papert (1990), Vygotsky (1978), and Piaget (see Piaget and Cook, 1954), as well as 21st-century theorists, such as Kafai (2018) and Resnick (2012). The sociocultural lens may overcome some of the individualistic connotations of coding by empowering researchers to examine knowledge as socially constructed, where students learn coding as one learns a new language—through dynamic interactions with self, others, technology, and culture. This approach integrates the role of others in learning as pillars of the learning experience, particularly through guided participation

(e.g., "scaffolding"), participatory appropriation, and apprenticeship by a more knowledgeable other (Rogoff, 2008).

Within the sociocultural-oriented studies, it was not just "making games" that exemplified a sociocultural framework, but also in "making games for and with other students" (Denner et al., 2012, p. 241). It is thus unsurprising that the sociocultural theories, including constructivist and social-constructivist perspectives, have been closely aligned with what is known as the "maker movement" (Bass et al., 2016; Hughes & Morrison, 2018; Sheridan et al., 2014; Tofel-Grehl et al., 2021). This approach seeks to expand the curiosity of learners by guiding them through a "tinkering" process as a means for making (e.g., robots, video games, computers), leading to "rich opportunities to reimagine what is taught and what is possible" (Tofel-Grehl et al., 2021, pp. 120, 127). Individual identity within a sociocultural coding framework is not considered apart from the group identity, but a vital component of it. Students are encouraged to pursue learning of personal interests within a supportive community, directing their skills for academic and career purposes (Ito et al., 2013). A sociocultural lens does not necessarily exclude traditional approaches, such as explicit teaching and direct instruction (Barak & Zadok, 2009).

Applying a Student-Centered Lens to Understanding Coding and Computational Thinking

A potential limitation of the sociocultural lens is that it may not cater to situations where learning is led by and for students, taking into account their individual differences, learning needs, strengths, and weaknesses. Such approaches to coding, sometimes referred to as "student-centered approaches," were influential in 24.5% of articles (12 out of 49), and included themes such as student-centered design, flexible learning, participatory learning, student voice, learner agency, adaptive learning, and self-directed learning. According to McCombs (2001), student-centered approaches are characterized by four key features: including students in key decisions about their learning, valuing students' unique perspectives, respecting individual differences (e.g., in abilities, backgrounds, experiences), and situating students as cocreators of their own teaching and learning experiences (see p. 186).

Unlike sociocultural theories that can sometimes emphasize top-down knowledge gradients from expert to learner (e.g., apprenticeship, mentoring), student-centered approaches emphasize active participation, power sharing, flexibly adapting to learner needs, promoting autonomy, making content personally relevant, and using formative assessment to address learning needs and academic goals (see Bremner et al., 2022). Parry et al. (2020), for example, applied a student-centered approach to coding known as a "third space" in which the production of video games is used not just as a space to develop coding skills but to promote agency and voice among students, develop critical and creative thinking in personally meaningful contexts, disrupt conventional narratives, and challenge hegemonies. They draw on the notion of the third space as contested, negotiated, and inherently political (McDougall & Potter, 2019). Others have found student-centered approaches to coding and computational thinking provide insights into promoting equity among students with disabilities in STEM fields (Lindsay & Hounsell, 2017) and for the teaching of complex content (Tofel-Grehl et al., 2021).

Student-centered approaches are not without criticism. For example, Bremner et al. (2022) explored student-centered approaches revealing a paucity of evidence of their effectiveness on objective learning outcomes, few longitudinal studies to demonstrate efficacy over time, and an overreliance on computer-mediated supports that widen the gap between the technology "haves" and "have nots." Critics have also accused student-centered approaches of promoting active students and passive teachers, prioritizing individual experience over linguistically mediated cultural knowledge, and conflating educational outcomes with developmental processes (see Mascolo, 2009, p. 7). A tension between student- and teacher-centered approaches may arise when students have limited prior knowledge of programming (Denner et al., 2012). In such instances, teachers may benefit from momentarily adopting teacher-centered techniques. These shortcomings may explain why the authors in our reviewed studies favorably cited sociocultural (41.7%) over student-centered (24.5%) approaches.

Applying a Design Thinking Lens to Coding and Computational Thinking

It is difficult to consider the student-centered paradigm separate from the design elements of the learning task and environment. This highlights the importance of "design thinking," which can be characterized by "an analytic and creative process that engages a person in opportunities to experiment, create and prototype models, gather feedback, and redesign" (Razzouk & Shute, 2012, p. 330). In our synthesis, we identified 22.44% of articles (11 out of 49) that employed this framework in coding classrooms. It was often closely coupled with the notion of "authentic learning" that emphasizes cultural relevance, authentic learning pedagogy, real-world relevance, lifelong learning, and aims to foster interest-driven participation among students.

Design thinking may also be associated with themes of instructional design, studio design, architecture, aesthetics, arts design, games-based design, and scaffolding. A "studio model" of pedagogy, for example, offers a design-thinking approach that is based on (a) complex, authentic, real-world projects characterized by "ill-defined problems"; (b) guided problem solving with creative constraints that foster "early failure and later success"; and (c) externalization and reflection—whereby learning artefacts are produced and elicit discussion (see Fields et al., 2015, pp. 615–616). Gaeta et al. (2019) have suggested that design-driven approaches may retain a distinct pedagogical purpose by adopting applications (e.g., Pocket Code) that are codesigned by teachers and students to better reflect specific learning and teaching needs.

Applying a Digital Literacies Lens to Coding and Computational Thinking

Digital literacies are vital for students to navigate an information-rich digital media environment. For this reason, digital literacies extend beyond mere technical skills (e.g., typing, web searching), enabling students to engage in multiple modes of learning; develop their digital identities; make meaning of the world; build digital records of past experience; and improve confidence, self-efficacy, and motivation—all within and beyond the context of the digital environments where such practices take place (Littlejohn et al., 2012). The notion of digital literacies in the context of our review is therefore situated within a broader context

of "collaboratively accomplished social practices that are achieved through human and non-human elements" (see Mills, 2016, p. 124).

A digital literacies lens was emphasized in 8 of the 49 articles (16.3%). Everyday digital literacies, digital media literacies, new literacies, multimodal texts, digital storytelling, production pedagogies, and coding as a new literacy, were commonly used terms—alongside the notion of digital literacies—though each with nuanced distinctions. A recurrent theme was that digital literacies are both "meaningful and generative of meaning" for the learner (Littlejohn et al., 2012, p. 551). Ascenzi-Moreno et al. (2020) applied this in a coding study that not only engaged and honored students' cultural and linguistic diversity but enabled them to "participate in and create new ways of engaging in literacy" (p. 51). Others argue that a digital literacies lens can empower culturally diverse students to critique media affordances and have space to experiment and play (Bers et al., 2019; Price & Price-Mohr, 2018).

Applying a Socially Inclusive Lens to Coding and Computational Thinking

We identified socially inclusive approaches in 20.4% of articles (10 out of 49) where programs targeted traditionally marginalized groups such as students with disabilities, those enrolled in special education and gifted programs, Indigenous and ethnically diverse students, as well as those from lower-socioeconomic backgrounds. For example, a case analysis by Snodgrass et al. (2016) applied a Universal Design for Learning framework to improve computational thinking among students with disabilities using Scratch and Code.org. Others have aimed their efforts at examining ways of differentiating teaching and learning to better support coding and computational thinking among Indigenous, gifted, talented, and creative students (Hagge, 2017; Hughes & Morrison, 2018; Tzou et al., 2019), although such efforts are not without challenges. For example, teachers expressed concerns about the considerable time, effort, and expertise needed to develop and maintain bespoke pedagogies that maximize inclusion of diverse learners (see Snodgrass et al., 2016).

Discussion

The development of new coding and computational thinking curricula and pedagogies to solve cross-disciplinary problems are needed to augment students' education in preparation for impending digital disruption to the future workforce. Extending well beyond computing and writing code, computational thinking involves essential multidisciplinary skillsets that are essential to future careers in a digital knowledge economy at the intersection of advanced human cognition and computing. This systematic review has brought together a substantial body of original studies demonstrating that computer coding is no longer conceived and taught as a niche technical skill of computer scientists, but as transverse and critical computational thinking skills—complex and powerful ways of reasoning and solving problems through mathematics and writing (Papert, 1990, p. 7). The requirement for such ways of thinking to be taught in schooling and postsecondary education is heightened as educators prepare students for digital participation and future workforces, where digital divides lead to an intensification of economic inequities amid a technological revolution.

In the majority of studies within the meta-synthesis, the gender of participants was described, but gender differences in coding experiences were not a specific topic of interrogation. Of those studies within the meta-synthesis that reported the gender composition of their cohorts, there were fewer female participants overall (male = 76.1%; female = 23.9%), and binary classifications of gender predominate. This is relevant given concerns about girls' lower levels of engagement in computing in the past (Singh et al., 2007). Gender has been a key variable in several quantitative studies investigating, for example, skills evaluation and opinion data (e.g., Atmatzidou & Demetriadis, 2016; Choi 2018). However, qualitative research has the potential for broader and deeper interrogation of gendered experiences of coding and computational thinking when taught across the curriculum.

RQ1—Designing Successful Coding and Computational Thinking Environments

A salient feature of successful coding and computational thinking environments was the need to build and mobilize relationships of support in communities of practice through the sustained and distributed expertise of peers as capable novices in online, offline, informal, and formal learning contexts. This distribution includes shared tasks, ideas, skills, knowledge, and thinking strategies within and beyond education to include community, industry, and interdisciplinary learning contexts where students are encouraged to ask questions, receive help, and assist others (Bass et al., 2016; Calabrese Barton & Tan, 2018; Fessakis et al., 2013; Gestwicki & McNely, 2012; Israel et al., 2015; Kafai et al., 2010; Ward et al., 2014). The role of communities of practice has not always been visible in previous studies and reviews that focus primarily on activities, tools, or platforms for skills training and performance (Li et al., 2020).

Student-driven learning in contexts where traditional instruction and hierarchical power were reorganized, but not absent, provided greater opportunities for students to demonstrate responsibility and autonomy (Calabrese Barton & Tan, 2018; Grubbs, 2013; Israel et al., 2015; Kafai et al., 2010; Sheridan et al., 2014), while also requiring timely explicit instruction across all age groups, particularly for high-level programming and efficient coding (Barak & Zadok 2009; Bargagna et al., 2019; Denner et al., 2012; Franetovic, 2016). These contexts in which students' ownership, choice, and agency were prioritized, were characterized by other vital characteristics, such as adequate teacher expertise (Khanlari, 2016), teacher questioning skills (Israel et al., 2015), sufficient time for coding (Calabrese Barton & Tan, 2018; Fessakis et al., 2013), strategic time management, technical support, tools for making, and well-organized materials and resources (Egbert et al., 2021; Grubbs, 2013; Ward et al., 2014).

RQ2—Evidence of Educational Outcomes for Coding and Computational Thinking

The educational outcomes for students engaged in coding integrated across the curriculum encompassed four key areas: (a) computational thinking, problem solving, creativity, and critical thinking; (b) disciplinary knowledge through coding; (c) student agency, self-direction, motivation, and active engagement; and (d) social and collaborative skills and taking diverse perspectives. The findings on

student outcomes point to the efficacy of integrating coding to advance computational thinking and problem solving for students across a range of ages, starting in early childhood classrooms with 3- to 5-year-olds (Bass et al., 2016; Bers et al., 2019; Egbert et al., 2021; Falloon, 2016; Fessakis et al., 2013). Many of the studies positioned computational thinking, problem solving, creativity, and critical thinking within the frame of the STEM curriculum, pointing to a gap identified by others (e.g., Kite et al., 2021; Li et al., 2020), in research related to broader application across the curriculum. Coding to solve problems across a range of fields beyond STEM requires interdisciplinary understanding—to create simulated learning environments, explain difficult concepts, and navigate virtual and real worlds.

Research on developing coding and computational thinking across disciplines beyond STEM is emerging with a focus on architecture, civic education, economics, English, e-textiles, history, humanities, media, music, and social studies (e.g., An, 2016; Gestwicki & McNely, 2012; Literat & Kligler-Vilenchik, 2018; Ratcliff & Anderson, 2011; Sáez-López et al., 2016). Coding experiences created opportunities to advance participatory cultures, promote interest-driven, build dynamic spaces for communities (Fields et al., 2015; Hughes & Morrison, 2018), and advance students' interpersonal skills and communication (Hagge, 2017). While teaching "perspective taking" can be challenging for teachers, students can learn this skill when processing constructive and peer feedback through testing iterations of designs (An, 2016; Bass et al., 2016).

RQ3—Managing Common Constraints for Teaching Coding and Computational Thinking

A range of pedagogical constraints frequently arise in coding and computational thinking learning contexts. Lack of teacher confidence negatively impacts teachers' capacity to help advance students' coding and computational thinking (Denner et al., 2012; DeVane et al., 2016; Egbert et al., 2021; Rich et al., 2021; Snodgrass et al., 2016). General technological and specific coding software difficulties can impede the development of student coding abilities, and the range and efficiency of coding techniques applied by students (Baytak & Land, 2010; Egbert et al., 2021; Frydenberg, 2015; Gaeta et al., 2019; Kafai et al., 2010, 2014). Student engagement in online coding communities, such as Scratch, highlights the need for moderation and scaffolding of expectations and interactions, particularly for younger students (Brennan, 2016; Burke, 2012; Franetovic, 2016; Kafai et al., 2010).

Cognitive-based difficulties associated with students' inability to complete complex coding tasks, combined with whether teachers offer insufficient or too much support, impacts students' coding proficiency, engagement, and motivation (Bargagna et al., 2019; Denner et al., 2012; Falloon, 2016; Hughes & Morrison, 2018; Ratcliff & Anderson, 2011). Other constraints resonate with findings by Ogegbo and Ramnarain (2022) who illustrate how a paucity of professional learning time and resource constraints can be challenging. Current findings further highlighted timing of lessons, strict teaching styles, difficult peer relationships, and lack or excessive reliance on programming materials (Bers et al., 2019; Denner et al., 2012; Egbert et al., 2021; Franetovic, 2016; Hughes & Morrison,

2018; Rich et al., 2021; Tonbuloğlu & Tonbuloğlu, 2019; Ward et al., 2014). Limited time in the curriculum was cited as a limitation for enabling students to develop higher-order thinking and coding-related skills, such as problem solving, or for fully realizing intentions for coded projects (Baytak & Land, 2010; Egbert et al., 2021; Israel et al., 2015; Tonbuloğlu & Tonbuloğlu, 2019).

RQ4—Gaps in Conceptual Frameworks for Coding and Computational Thinking

The field of coding and computational thinking remains dynamic and contested. Sociocultural and computational thinking approaches were represented in over half of the articles, with conceptual frameworks often poorly delineated. Studies often claimed to adopt a specific conceptual paradigm, but seldom reflected its principles consistently in their study design, methodology, and interpretation of data, while there is a lack of common theoretical frameworks for conceptualizing computational thinking (Kite et al., 2021; Li et al., 2020).

Only one study examined coding in an exclusively online context; all other studies were conducted either partly or entirely in face-to-face contexts, making the theorizing of online coding studies a relatively neglected area of research (Aragon et al., 2009). This is important, given the concerted effort by major educational institutes toward cultivating digital literacies in online communities (see Organization for Economic Cooperation and Development [OECD], 2021). The global COVID-19 pandemic of the 2020s temporarily restricted the face-to-face teaching of coding and computational thinking, highlighting the need for online pedagogies.

Future Directions

A key recommendation for the future teaching of problem solving and computational thinking skills is to introduce these early in a students' learning trajectory (Bers et al., 2019; Buitrago Flórez et al., 2017; Fessakis et al., 2013), given the time needed for the cumulative development of debugging skills and higher-level computational thinking. Short interventions beginning late in students' schooling do not position them for long-term success (Bass et al., 2016). Additionally, problem-solving skills in coding are effectively taught when students are encouraged to build perseverance with debugging challenges, while cross-disciplinary approaches can support the development of high-level thinking skills using integrated curricula with real-world applications (Falloon, 2016).

Student engagement in online communities comes with recommendations for age-appropriate useability, moderation, structures, and social support to manage and cultivate positive interactions (Brennan, 2016; Burke, 2012; Fields et al., 2015; Hagge, 2017; Kafai et al., 2010, 2012; Literat & Kligler-Vilenchik, 2018; Vasudevan et al., 2015). This is because supportive online coding communities, including those external to the classroom, are increasingly used to broaden students' access to networked support, remix code, act on useful feedback, and receive recognition for their contributions as members of larger communities of practice.

A key point in relation to learning outcomes across the curriculum (RQ2) is to further realize the benefits of teaching coding beyond STEM related

studies—though there were exciting examples of approaches beginning to take up this challenge. Further integration of coding across the curriculum more broadly may require systematic professional development for teachers, and more ready access to technology resources and support. Future research for advancing coding research in education will need longer-term interventions and longitudinal studies that apply the teaching of coding across varied subject areas with cumulative skills development and sufficient time in the curriculum (Egbert et al., 2021; Gaeta et al., 2019; Khanlari, 2016; Rich et al., 2021; Tonbuloğlu & Tonbuloğlu, 2019).

To overcome barriers to teaching coding beyond the STEM curriculum (RQ3), cross-curriculum design needs to prioritize the integration of deep academic content through programming, rather than shallow applications of existing content area knowledge (An, 2016). Projects should allow students sufficient time to read and research new content-area knowledge to apply to their game designs in ways that are not limited by their technical proficiencies.

There is still a paucity of well-conceptualized studies that provide concrete practices for K–12 educators to explicitly link coding and computational thinking to core content of subject areas (Hickmott et al., 2018). Strategies to strengthen the quality and duration of both pre-service and in-service K–12 teacher training in coding and computational thinking should be a key priority to harness students' cultural capital of computational thinking—the values, skills, and dispositions that provide the cultural building material for mature problem-solving and digital skills. Additionally, educational organizations need to remove barriers to developing student and teacher coding proficiency by investing in sufficient technology resources and technical support personnel, including more extensive professional development and in-class support for teachers of coding and computational thinking, particularly for those who are not primarily trained in technology or STEM.

Future directions for research of coding and computational thinking in education should address the rapidly developing impact of AI. Developers now use AI code generators to rapidly create functional code, allowing more time for creative thinking and problem solving, while coding remains essential to program, fine-tune, and implement algorithms and models. New research of AI-based coding is needed to teach students to understand how AI works, including providing students with opportunities with programs like Scratch to use, create, and modify the partial coding of algorithms to explore and critique the behavior, biases, and ethics of machine learning algorithms and the underlying "computational thinking" embedded in AI processes (Estevez et al., 2019).

Conclusion

The unprecedented growth of the teaching of coding and computational thinking—both in schools and in out-of-school contexts—and the urgent need for the development of computational ways of thinking is not the exclusive concern of the technology teacher but the concern of all in education. The evidence consistently points to the teaching of coding and computational thinking as skills that should be taught alongside reading, writing, and arithmetic (Wing, 2006, p. 33)—a view that is supported by countries that value

computational thinking in the core curriculum, such as the United States, UK, Finland, Australia, and New Zealand (Uzunboylu et al., 2017).

This meta-synthesis has demonstrated that teaching new ways of thinking through computational practices needs to begin early in a child's education (Bass et al., 2016; Buitrago Flórez et al., 2017), and continue into postsecondary (Bers et al., 2019; Fessakis et al., 2013). The recent global pandemic has heightened the need for universal access to digital technologies and skill sets to prepare students for participation in a world economy that is now heavily reliant on problem solving with digital technologies.

The 49 studies that met the criteria for inclusion were systematically analyzed to identify the essential features of learning environments found to support the development of students' coding and computational thinking most successfully, including when taught across the curriculum. The findings point to future directions whereby schools need to mobilize relationships of support for learning to code within communities of practice through distributed expertise and help-seeking behavior in both online and offline learning contexts. Expert-novice interaction should occur at strategic points, such as peer playtesting games and providing constructive feedback, or to model, remix, or refine shared code (e.g., Baytak & Land, 2010; Sáez-López et al., 2016; Tzou et al., 2019). As AI transforms coding, human-centric critical thinking through distributed expertise will be vital to solve key questions in the future with human values and ethics to guide machines.

Beginning the teaching of coding and computational thinking from early childhood, student-driven learning in contexts to build students' perseverance with coding is the predominant approach. However, learning environments must be structured optimally to provide teacher or expert instruction or guidance for higher-level coding (Barak & Zadok, 2009). Teacher support is necessary to manage time limitations and prevent student frustration, particularly with longer-term projects (e.g., Denner et al., 2012; Egbert et al., 2021). Connecting to online communities does not guarantee positive experiences for all but, when utilized, must be managed in a guided and reflective way appropriate to students' confidence to share their work and to act on constructive feedback (Literat & Kligler-Vilenchik, 2018).

Recent studies have researched the teaching of code beyond the context of STEM classrooms—although science, technology, and mathematics are still the dominant contexts—demonstrating the potential to use coding as a springboard for teaching a wide range of valuable cross-curricular learning outcomes. While prioritizing the successful teaching of coding and computational thinking as aims in themselves, programs have targeted and measured students' significant improvement in problem solving, critical thinking, student agency, self-direction, motivation, active engagement, creativity, writing, self-reflexivity, perspective taking, and social and collaborative skills (e.g., Bass et al., 2016; Fields et al., 2015; Hagge, 2017; Hughes & Morrison, 2018). Further integration of coding across the curriculum beyond STEM requires the systematic professional development of teachers and pre-service teachers that extends beyond one semester, with more ready access to technology hardware and software, technical personnel or partnerships, and knowledge of how to connect coding with subject matter (Khanlari, 2016).

Importantly, mandated curricula and assessment in many countries are not keeping up with the changes in technology, creating a misalignment between the rich learning in coding and computational thinking programs that is currently not measured in mandated assessments, creating a barrier for teachers to include robotics and game design in non-STEM subjects (Khanlari, 2016). In terms of advancing coding research in education, longer-term interventions and longitudinal studies are needed that apply the teaching of coding across varied subject areas (Egbert et al., 2021; Rich et al. 2021; Tonbuloğlu & Tonbuloğlu, 2019). As society embraces Industry 5.0, calling for highly skilled people with problem-solving expertise, this domain will undoubtedly continue to be an asset to our economic future.

Supporting the view among educators, economists, governments, and nations that computer programming and computational thinking are complex, definitive, and transversal skills that children should learn in school, this review provides evidence that coding can be successfully taught to consolidate knowledge and skills across the curriculum, from mathematics to history, architecture to civic education, and from English and to the arts, while highlighting specific challenges that need to be overcome in these endeavors (e.g., An, 2016; Literat & Kligler-Vilenchik, 2018; Sáez-López et al., 2016). The teaching of computational thinking extends beyond computing and writing code and is now essential for full participation in a democratic digital society amidst an AI-based technological revolution of the Second Machine Age (Mills, 2023), where complex skills are required to solve intellectually engaging, authentic, real-world problems that are not bounded by disciplinary borders.

Funding Information

This research was supported in part by the Australian Research Council's Discovery Projects funding scheme (DP190100228 and FT180100009). The views expressed herein are those of the authors and are not necessarily those of the Australian Research Council.

ORCID iDs

Kathy A. Mills https://orcid.org/0000-0003-1140-3545
Jen Cope https://orcid.org/0000-0001-7003-4474
Laura Scholes https://orcid.org/0000-0002-8849-2825
Luke Rowe https://orcid.org/0000-0001-5027-733X

References

References marked with an asterisk (*) indicate articles included in the meta-synthesis.

Abrami, P. C., Bernard, R. M., Borokhovski, E., Waddington, D. I., Wade, C. A., & Persson, T. (2015). Strategies for teaching students to think critically: A meta-analysis. *Review of Educational Research*, *85*(2), 275–314. https://doi.org/10.3102/0034654314551063

Alexander, P. A. (2020). Methodological guidance paper: The art and science of quality systematic reviews. *Review of Educational Research*, *90*, 6–23. https://doi.org/10.3102/0034654319854352

*An, Y. (2016). A case study of educational computer game design by middle school students. *Education Technology Research and Development*, *64*, 555–571. https://doi.org/10.1007/s11423-016-9428-7

*Aragon, C. R., Poon, S. S., Monroy-Hernández, A., & Aragon, D. (2009, October). A tale of two online communities: Fostering collaboration and creativity in scientists and children. In *Proceedings of the Creativity and Cognition Conference* (pp. 9–18). ACM Press. https://doi.org/10.1145/1640233.1640239

*Ascenzi-Moreno, L., Güílamo, A., & Vogel, S. E. (2020). Integrating coding and language arts: A view into sixth graders' multimodal and multilingual learning. *Voices From the Middle*, *27*(4), 47–52. https://par.nsf.gov/biblio/10186626

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: *Robotics and Autonomous Systems*, *75*, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

*Barak, M., & Zadok, Y. (2009). Robotics projects and learning concepts in science, technology and problem solving. *International Journal of Technology and Design Education*, *19*(3), 289–307. https://doi.org/10.1007/s10798-007-9043-3

*Bargagna, S., Castro, E., Cecchi, F., Cioni, G., Dario, P., Dell'Omo, M., Di Lieto, M. C., Inguaggiato, E., Martinelli, A., Peci, C., & Sgandurra, G. (2019). Educational robotics in Down Syndrome: A feasibility study. *Technology, Knowledge and Learning*, *24*(2), 315–323. https://doi.org/10.1007/s10758-018-9366-z

*Bass, K. M., Hu Dahl, I., & Panahandeh, S. (2016). Designing the game: How a project-based media production program approaches STEAM career readiness for underrepresented young adults. *Journal of Science Education and Technology*, *25*(6), 1009–1024. https://doi.org/10.1007/s10956-016-9631-7

*Baytak, A., & Land, S. M. (2010). A case study of educational game design *by* kids and *for* kids. *Procedia-Social and Behavioral Sciences*, *2*(2), 5242–5246. https://doi.org/10.1016/j.sbspro.2010.03.853

*Bers, M. U., González-González, C., & Armas–Torres, M. B. (2019). Coding as a playground: Promoting positive learning experiences in childhood classrooms. *Computers & Education*, *138*, 130–145. https://doi.org/10.1016/j.compedu.2019.04.013

Bremner, N., Sakata, N., & Cameron, L. (2022). The outcomes of learner-centred pedagogy: A systematic review. *International Journal of Educational Development*, *94*, 102649. https://doi.org/10.1016/j.ijedudev.2022.102649

*Brennan, K. (2016). Audience in the service of learning: How kids negotiate attention in an online community of interactive media designers. *Learning, Media and Technology*, *41*(2), 193–212. https://doi.org/10.1080/17439884.2014.939194

Brennan, K., & Resnick, M. (2012, April 13–17). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association* (*Vol. 1*, p. 25), Vancouver.

Brown, A. L., Ash, D., Rutherford, M., Nakagawa, K., Gordon, A., & Campione, J. C. (1993). Distributed expertise in the classroom. In G. Salomon, *Distributed cognitions: Psychological and educational considerations* (pp.188–228). CUP.

Buitrago Flórez, F., 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. https://doi.org/10.3102/0034654317710096

*Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education*, *4*(2), 121–135.

*Calabrese Barton, A., & Tan, E. (2018). A longitudinal study of equity-oriented STEM-rich making among youth from historically marginalized communities. *American Educational Research Journal*, *55*(4), 761–800. https://doi.org/10.3102/0002831218758668

Choi, K. S. (2015). A comparative analysis of different gender pair combinations in pair programming. *Behaviour & Information Technology*, *34*(8), 825–837.

Cooper, H., Hedges, L., & Valentine, J. (Eds.). (2009). *The handbook of research synthesis and meta-analysis* (2nd ed.). Russell Sage Foundation.

Deloitte Access Economics. (2017). *Australia's digital pulse policy: Priorities to fuel Australia's digital workforce boom*. Australian Computer Society.

*Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, *58*(1), 240–249. https://doi.org/10.1016/j.compedu.2011.08.006

*DeVane, B., Steward, C., & Tran, K. M. (2016). Balancing expression and structure in game design: Developing computational participation using studio-based design pedagogy. *Educational Technology*, *56*(3), 42–47. https://www.jstor.org/stable/44430492

*Egbert, J., Shahrokni, S. A., Abobaker, R., & Borysenko, N. (2021). "It's a chance to make mistakes": Processes and outcomes of coding in 2nd grade classrooms. *Computers & Education*, *168*, 104173 https://doi.org/10.1016/j.compedu.2021.104173

Estevez, J., Garate, G., & Graña, M. (2019). Gentle introduction to artificial intelligence for high-school students using scratch. *IEEE Access*, *7*, 179027–179036.

*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. https://doi.org/10.1111/jcal.12155

*Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87–97. https://doi.org/10.1016/j.compedu.2012.11.016

Feurzeig, W., Papert, S. A., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments*, *19*(5), 487–501. https://doi.org/10.1080/10494820903520040

*Fields, D., Vasudevan, V., & Kafai, Y. B. (2015). The programmers' collective: Fostering participatory culture by making music videos in a high school Scratch coding workshop. *Interactive Learning Environments*, *23*(5), 613–633. https://doi.org/10.1080/10494820.2015.1065892

*Franetovic, M. (2016). Future game developers within a virtual world: Learner archetypes and team leader attributes. *Journal of Educational Multimedia and Hypermedia*, *25*(4), 343–361. https://www.learntechlib.org/primary/p/173244/

*Frydenberg, M. (2015). Achieving digital literacy through game development: An authentic learning experience. *Interactive Technology and Smart Education*, *12*(4), 256–269. https://doi.org/10.1108/ITSE-08-2015-0022

*Gaeta, E., Beltrán-Jaunsaras, M. E., Cea, G., Spieler, B., Burton, A., García-Betances, R. I., Cabrera-Umpiérrez, M. F., Brown, D., Boulton, H., & Arredondo Waldmeyer, M. T. (2019). Evaluation of the Create@School game-based learning–teaching approach. *Sensors*, *19*(15), 3251. https://doi.org/10.3390/s19153251

Gee, J. P. (2003). What video games have to teach us about learning and literacy. *Computers in Entertainment*, *1*(1), 20–20. https://doi.org/10.1145/950566.950595

*Gestwicki, P., & McNely, B. (2012, October). A case study of a five-step design thinking process in educational museum game design. In Proceedings of Meaningful Play 2012 Conference. Michigan State University. http://meaningfulplay.msu.edu/proceedings2012/mp2012_submission_37.pdf

Grimes, S. M., & Fields, D. A. (2015). Children's media making, but not sharing: The potential and limitations of child-specific DIY media websites. *Media International Australia*, *154*(1), 112–122. https://doi.org/10.1177%2F1329878X1515400114

*Grubbs, M. (2013). Robotics intrigue middle school students and build STEM skills. *Technology and Engineering Teacher*, *72*(6), 12–16.

Gutiérrez, K. D. (2008). Developing a sociocritical literacy in the third space. *Reading Research Quarterly*, *43*(2), 148–164. https://doi.org/10.1598/RRQ.43.2.3

*Hagge, J. (2017). Scratching beyond the surface of literacy: Programming for early adolescent gifted students. *Gifted Child Today*, *40*(3), 154–162. https://doi.org/10.1177/1076217517707233

Hickmott, D., Prieto-Rodriguez, E., & Homes, K. (2018). A scoping review of studies on computational thinking in K–12 mathematics classrooms. *Digital Experiences in Mathematics Education*, *4*(1), 48–69. https://doi.org/10.1007/s40751-017-0038-8

*Hughes, J., & Morrison, L. (2018). The use of e-textiles in Ontario education. *Canadian Journal of Education*, *41*(1), 356–384. https://journals.sfu.ca/cje/index.php/cje-rce/article/download/3237/2495/0

*Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, *82*, 263–279. https://doi.org/10.1016/j.compedu.2014.11.022

Ito, M., Gutiérrez, K., Livingstone, S., Penuel, B., Rhodes, J., Salen, K., Schor, J., Sefton-Green, J., & Watkins, S. C. (2013) *Connected learning: An agenda for research and design*. Digital Media and Learning Research Hub. http://eprints.lse.ac.uk/id/eprint/48114

Kafai, Y. B. (2018). Constructionist visions: Hard fun with serious games. *International Journal of Child-Computer Interaction*, *18*, 19–21. https://doi.org/10.1016/j.ijcci.2018.04.002

*Kafai, Y. B., Fields, D., & Burke, W. Q. (2010). Entering the clubhouse: Case studies of young programmers joining the online Scratch communities. *Journal of Organizational and End User Computing*, *22*(2), 21–35. http://doi.org/10.4018/joeuc.2010101906

*Kafai, Y. B., Fields, D. A., Roque, R., Burke, W. Q., & Monroy-Hernandez, A. (2012). Collaborative agency in youth online and offline creative production in Scratch. *Research and Practice in Technology Enhanced Learning*, *7*(2), 63–87. https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1175&context=itls_facpub

*Kafai, Y., Lee, E., Searle, K., Fields, D., Kaplan, E., & Lui, D. (2014). A crafts-oriented approach to computing in high school: Introducing computational concepts, practices, and perspectives with electronic textiles. *ACM Transactions on Computing Education (TOCE)*, *14*(1), 1–20. https://doi.org/10.1145/2576874

*Kafai, Y. B., & Vasudevan, V. (2015, June). Hi-Lo tech games: Crafting, coding, and collaboration of augmented board games by high school youth. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 130–139). https://doi.org/10.1145/2771839.2771853

*Khalili, N., Sheridan, K., Williams, A., Clark, K., & Stegman, M. (2011). Students designing video games about immunology: Insights for science learning. *Computers in the Schools*, *28*(3), 228–240. https://doi.org/10.1080/07380569.2011.594988

*Khanlari, A. (2016). Teachers' perceptions of the benefits and the challenges of integrating educational robots into primary/elementary curricula. *European Journal of Engineering Education*, *41*(3), 320–330. https://doi.org/10.1080/03043797.2015.1056106

Kite, V., Park, S., & Wiebe, E. (2021). The code-centric nature of computational thinking education: A review of trends and issues in computational thinking education research. *Sage Open*, *11*(2), 21582440211016418. https://doi.org/10.1177/21582440211016418

Li, Y., Schoenfeld, A. H., diSessa, A. A., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020). On computational thinking and STEM education. *Journal for STEM Education Research*, *3*, 147–166. https://doi.org/10.1007/s41979-020-00044-w

*Lindsay, S., & Hounsell, K. G. (2017). Adapting a robotics program to enhance participation and interest in STEM among children with disabilities: A pilot study. *Disability and Rehabilitation: Assistive Technology*, *12*(7), 694–704. https://doi.org/10.1080/17483107.2016.1229047

*Literat, I., & Kligler-Vilenchik, N. (2018). Youth online political expression in non-political spaces: Implications for civic education. *Learning, Media, and Technology*, *43*(4), 400–417. https://doi.org/10.1080/17439884.2018.1504789

Littlejohn, A., Beetham, H., & McGill, L. (2012). Learning at the digital frontier: A review of digital literacies in theory and practice. *Journal of Computer Assisted Learning*, *28*(6), 547–556. https://doi.org/10.1111/j.1365-2729.2011.00474.x

Lockwood, J., & Mooney, A. (2017). *Computational thinking in education: Where does it fit? A systematic literary review*. Cornell University. https://doi.org/10.48550/arXiv.1703.07659

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. https://doi.org/10.1016/j.chb.2014.09.012

Martin, C. (2017). Expressing youth voice through video games and coding, *Knowledge Quest*, *45*(4), 50–57.

Mascolo, M. F. (2009). Beyond student-centered and teacher-centered pedagogy: Teaching and learning as guided participation. *Pedagogy and the Human Sciences*, *1*(1), 3–27.

McCombs, B. L. (2001). What do we know about learners and learning? The learner-centered framework: Bringing the educational system into balance. *Educational Horizons*, *79*(4), 182–193. https://www.jstor.org/stable/42927064

McDougall, J., & Potter, J. (2019). Digital media learning in the third space. *Media Practice and Education*, *20*(1), 1–11. https://doi.org/10.1080/25741136.2018.1511362

Mercer, N., & Howe, C. (2012). Explaining the dialogic processes of teaching and learning: The value and potential of sociocultural theory. *Learning, Culture and Social Interaction*, *1*(1), 12–21. https://doi.org/10.1016/j.lcsi.2012.03.001

Mills, K. A. (2016). *Literacy theories for the digital age: Social, critical, multimodal, spatial, material and sensory lenses*. Multilingual Matters.

Mills, K. A. (2023). Critical literacy and social media for L1 language learners in the second machine age. *Danish Journal of Knowledge About Literacy (Viden om Literacy)*, *34*, 48–55. https://videnomlaesning.dk/media/5645/kathy-a-mills.pdf

Ogegbo, A. A., & Ramnarain, U. (2022). Teachers' perceptions of and concerns about integrating computational thinking into science teaching after a professional development activity. *African Journal of Research in Mathematics, Science and Technology Education*, *26*(3), 1–11. https://doi.org/10.1080/18117295.2022.2133739

Organization for Economic Cooperation and Development. (2021). *OECD digital education outlook 2021: Pushing the frontiers with artificial intelligence, blockchain and robots*. OECD Publishing. https://doi.org/10.1787/589b283f-en

*Ospennikova, E., Ershov, M., & Iljin, I. (2015). Educational robotics as an innovative educational technology. *Procedia–Social and Behavioral Sciences*, *214*, 18–26. https://doi.org/10.1016/j.sbspro.2015.11.588

Papert, S. (1990). *A critique of technocentrism in thinking about the school of the future*. Cambridge Epistemology and Learning Group, MIT Media Laboratory.

Papert, S., & Harel, I. (1991). *Constructionism*. Ablex Publishing.

*Parry, R., Howard, F., & Penfold, L. (2020). Negotiated, contested and political: the disruptive third spaces of youth media production. *Learning, Media, and Technology*, *45*(4), 409–421. https://doi.org/10.1080/17439884.2020.1754238

Piaget, J., & Cook, M. (1954). *The construction of reality in the child*. Basic Books.

*Pinto-Llorente, A. M., Casillas-Martín, S., Cabezas-González, M., & García-Peñalvo, F. J. (2018). Building, coding, and programming 3D models via a visual programming environment. *Quality & Quantity*, *52*(6), 2455–2468. https://doi.org/10.1007/s11135-017-0509-4

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365–376. https://doi.org/10.1016/j.compedu.2018.10.005

*Price, C. B., & Price-Mohr, R. M. (2018). Stories children write while coding: a cross-disciplinary approach for the primary classroom. *Cambridge Journal of Education*, *48*(6), 735–747. https://doi.org/10.1080/0305764X.2017.1418834

*Ratcliff, C. C., & Anderson, S. E. (2011). Reviving the turtle: Exploring the use of Logo with students with mild disabilities. *Computers in the Schools*, *28*(3), 241–255. https://doi.org/10.1080/07380569.2011.594987

Razzouk, R., & Shute, V. (2012). What is design thinking and why is it important? *Review of Educational Research*, *82*(3), 330–348. https://doi.org/10.3102/0034654312457429

Resnick, M. (2012). Reviving Papert's dream. *Educational Technology*, *52*(4) 42–46. https://www.jstor.org/stable/44430058

*Rich, P. J., Mason, S. L., & O'Leary, J. (2021). Measuring the effect of continuous professional development on elementary teachers' self-efficacy to teach coding and computational thinking. *Computers & Education*, *168*, 104–196. https://doi.org/10.1016/j.compedu.2021.104196

Rogoff, B. (2008). Observing sociocultural activity on three planes: Participatory appropriation, guided participation, and apprenticeship. In K. Hall, P. Murphy, & J. Soler (Eds.), *Pedagogy and practice: Culture and identities* (pp. 58–74). Sage Publications.

*Sacristán, A. I., & Pretelín-Ricárdez, A. (2017). Gaining modelling and mathematical experience by constructing virtual sensory systems in maze-video games. *Teaching Mathematics and Its Applications: An International Journal of the IMA*, *36*(3), 151–166. https://doi.org/10.1093/teamat/hrw019

*Sáez-López, J. M., Román-González, M., & Vázquez-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. https://doi.org/10.1016/j.compedu.2016.03.003

Sandelowski, M., Voils, C. I., & Barroso, J. (2006). Defining and designing mixed research synthesis studies. *Research in the Schools*. *13*(1), 1–29.

Schlosser, R. W., Wendt, O., Bhavnani, S., & Nail-Chiwetalu, B. (2006). Use of information-seeking strategies for developing systematic reviews and engaging in evidence-based practice: The application of traditional and comprehensive pearl growing. A review. *International Journal of Language & Communication Disorders*, *41*(5), 567–582. https://doi.org/10.1080/13682820600742190

*Sheridan, K., Halverson, E., Litts, B., Brahms, L., Jacobs-Priebe, L., & Owens, T. (2014). Learning in the making: A comparative case study of three makerspaces. *Harvard Educational Review*, *84*(4), 505–531. https://doi.org/10.17763/haer.84.4.brr34733723j648u

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Shuval, K., Harker, K., Roudsari, B., Groce, N., Mills, B., Siddiqi, Z., & Shachak, A. (2011). Is qualitative research second class science? A quantitative longitudinal examination of qualitative research in medical journals. *PLoS one*, *6*(2), e16937. https://doi.org/10.1371/journal.pone.0016937

Singh, K., Allen, K. R., Scheckler, R., & Darlington, L. (2007). Women in computer-related majors: A critical synthesis of research and theory from 1994 to 2005. *Review of Educational Research*, *77*(4), 500–533. https://doi.org/10.3102/0034654307309919

*Snodgrass, M. R., Israel, M., & Reese, G. C. (2016). Instructional supports for students with disabilities in K-5 computing: Findings from a cross-case analysis. *Computers & Education*, *100*, 1–17. https://doi.org/10.1016/j.compedu.2016.04.011

Stockard, J., Wood, T. W., Coughlin, C., & Rasplica Khoury, C. (2018). The effectiveness of direct instruction curricula: A meta-analysis of a half century of research. *Review of Educational Research*, *88*(4), 479–507. https://doi.org/10.3102/0034654317751919

*Theodosiou, S., & Karasavvidis, I. (2015). Serious games design: A mapping of the problems novice game designers experience in designing games. *Journal of e-Learning and Knowledge Society*, *11*(3), 133–148. https://www.learntechlib.org/p/151929/

Thompson, R., Tanimoto, S., Lyman, R. D., Geselowitz, K., Begay, K. K., Nielsen, K., Nagy, W., Abbott, R., Raskind, M., & Berninger, V. (2018). Effective instruction for persisting dyslexia in upper grades. *Education and Information Technologies*, *23*(3), 1043–1068. https://doi.10.1007/s10639-017-9647-5

*Tofel-Grehl, C., Ball, D., & Searle, K. (2021). Making progress: Engaging maker education in science classrooms to develop a novel instructional metaphor for teaching electric potential. *Journal of Educational Research*, *114*(2), 119–129. https://doi.org/10.1080/00220671.2020.1838410

*Tonbuloğlu, B., & Tonbuloğlu, I. (2019). The effect of unplugged coding activities on computational thinking skills of middle school students. *Informatics in Education*, *18*(2), 403–426. https://doi.org/10.15388/infedu.2019.19

*Tzou, C., Meixi Suárez, E., Bell, P., LaBonte, D., Starks, E., & Bang, M. (2019). Storywork in STEM-Art: Making, materiality and robotics within everyday acts of Indigenous presence and resurgence. *Cognition and Instruction*, *37*(3), 306–326. https://doi.org/10.1080/07370008.2019.1624547

Uzunboylu, H., Kinik, E., & Kanbul, S. (2017). An analysis of countries which have integrated coding into their curricula and the content analysis of academic studies on coding training in Turkey. *TEM Journal*, *6*(4), 783–791. https://doi.org/10.18421/TEM64-18

*Vasudevan, V., Kafai, Y., & Yang, L. (2015, June). Make, wear, play: Remix designs of wearable controllers for scratch games by middle school youth. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 339–342). https://doi.org/10.1145/2771839.2771911

Vee, A. (2017). *Coding literacy: How computer programming is changing writing*. MIT Press.

Vygotsky, L. S. (1978). *Mind in society: The development of higher mental processes*. Harvard University Press.

*Ward, D., Hahn, J., & Mestre, L. (2014). Adventure code camp: library mobile design in the backcountry. *Information Technology and Libraries*, *33*(3), 45–52. https://doi.org/10.6017/ital.v33i3.5480

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wolfe, B. L., & Haveman, R. H. (2002, June). Social and non-market benefits from education in an advanced economy. In *Conference Series-Federal Reserve Bank of Boston*, *47*, 97–131. Federal Reserve Bank of Boston.

Authors

KATHY A. MILLS is a research professor at the Institute for Learning Science and Teacher Education, Australian Catholic University, Banyo Campus, QLD 4014; kathy.mills@acu.edu.au. She is a current future fellow of the Australian Research Council with four funded projects, including coding animated narratives and using extended reality technologies for multimodal design. Her research, including "A Review of the Digital Turn in the New Literacy Studies" (*RER*), addresses multimodal and digital literacies in education from diverse theoretical perspectives.

JEN COPE is a research manager at the Institute for Learning Sciences and Teacher Education, Australian Catholic University. She manages an Australian Research Council–funded project that develops pedagogies for students to code animated narratives, integrating multimodal authoring and computational thinking.

LAURA SCHOLES is a researcher and assistant professor at the Institute for Learning Science and Teacher Education, Australian Catholic University. She recently completed an Australian Research Council study investigating literacy pedagogies for primary classrooms. Her mixed methods research addresses key questions about student learning supported by technologies as educators embrace the digital age.

LUKE ROWE is a lecturer and researcher at the National School of Education, Australian Catholic University. Luke specializes in quantitative research methods and applies these skills to a diverse range of educational topics including meta-analyses, evidence-based teaching, the science of learning, digital literacies, and coding and computational thinking.