



ARTICLE



<https://doi.org/10.1057/s41599-025-04471-1>

OPEN

# LLM-based collaborative programming: impact on students' computational thinking and self-efficacy

Yi-Miao Yan<sup>1,2</sup>, Chuang-Qi Chen<sup>3</sup>, Yang-Bang Hu<sup>4</sup> & Xin-Dong Ye<sup>1,2</sup>

At present, collaborative programming is a prevalent approach in programming education, yet its effectiveness often falls short due to the varying levels of coding skills among team members. To address these challenges, Large Language Models (LLMs) can be introduced as a supportive tool to enhance both the efficiency and outcomes of collaborative programming. In this shift, the structure of collaborative teams evolves from human-to-human to a new paradigm consisting of human, human, and AI. To investigate the effectiveness of integrating LLMs into collaborative programming, this study designed a quasi-experiment. To explore the effectiveness of integrating LLMs into collaborative programming, we conducted a quasi-experiment involving 82 sixth- and seventh-grade students, who were randomly assigned to either an experimental group or a control group. The results showed that incorporating LLMs into collaborative programming significantly reduced students' cognitive load and improved their computational thinking skills. However, no significant difference in self-efficacy was observed between the two groups, likely due to the cognitive demand students faced when transitioning from graphical programming to text-based coding. Despite this, the study remains optimistic about the potential of LLM-enhanced collaborative programming, as students learning in this way exhibit lower cognitive load than those in conventional environments.

<sup>1</sup>College of Education, Wenzhou University, Wenzhou, China. <sup>2</sup>Key Research Center of Philosophy and Social Sciences of Zhejiang Province (Institute of Medical Humanities, Wenzhou Medical University), Wenzhou, China. <sup>3</sup>Institute of Language sciences, Shanghai International Studies University, Shanghai, China. <sup>4</sup>Center for Teacher Education Research, Beijing Normal University, Key Research Institute of the Ministry of Education, Beijing, China. email: [202131010052@mail.bnu.edu.cn](mailto:202131010052@mail.bnu.edu.cn); [yxd@wzu.edu.cn](mailto:yxd@wzu.edu.cn)

## Introduction

In the rapid development of the digital society, computational thinking has become an important component of human thinking (Tikva & Tambouris, 2021). Csizmadia proposes that all students should learn computational thinking (Csizmadia et al. 2015), regardless of discipline and age, if students have computational thinking skills, they will be able to utilize their creativity to solve problems and positively face the challenges brought by the society. Computational programming education has received much attention from scholars at both domestic and international level as a mainstream way to improve computational thinking. Scholars generally believe that one of the most effective ways to cultivate and develop students' computational thinking process skills is through computational programming education (Belmar 2022). Scholars pointed out that programming education has returned to primary and middle schools in recent years, computational thinking has become an important guiding principle in curriculum construction, and programming education has increasingly become an important way to develop students' problem-solving ability, creativity, and computational thinking (Tikva and Tambouris 2021). However, for novice programming, especially those who are exposed to text-based programming languages for the first time, they face many challenges (Louca and Zacharia 2008), such as unfamiliarity with syntax and structure, difficulty in organizing logic and algorithms, and difficulty in finding relevant resources and documentation. All these problems affect students' programming self-efficacy and cognitive load, which in turn affects students' learning effectiveness (Yildiz Durak 2018).

Collaborative Programming is an effective strategy in programming education to increase students' self-efficacy and computational thinking skills (Denner et al. 2014). However, the conventional collaborative programming teaching model often faces some difficulties in the implementation of the classroom (Strom and Strom 2002; Zheng and Zheng 2021), such as the common shortcomings and difficulties in the programming ability of the group members, which hinders the collaborative programming process and makes it difficult to achieve good results (Xinogalos et al. 2017). In addition, middle school students need to face the conversion from graphical programming to text-based programming, and the process of transitioning from figurative to abstract programming leads to a great challenge to students' self-efficacy (Mladenović et al. 2018).

In recent years, Artificial Intelligence Generated Content (AIGC) has attracted attention from many fields outside the computer science community, and large language models (LLMs) such as ChatGPT developed by OpenAI are one of the representative achievements of AIGC today (Cao et al. 2023). LLMs such as ChatGPT have the potential to improve several limitations and challenges of conventional collaborative models in teaching programming in primary and middle schools (Tan et al. 2023), such as heterogeneity issues, time and location constraints, feedback delays, and resource allocation issues. However, while the potential of LLMs is clear, empirical research on the integration of LLMs into K-12 programming education remains scarce (Zhang and Tur 2023).

Therefore, this study aims to explore the integration of LLMs in middle school collaborative programming context. By conducting LLM-supported collaborative programming instruction alongside traditional collaborative programming education, this study seeks to compare whether there are significant differences in their effectiveness in enhancing students' self-efficacy and computational thinking skills, and to provide some suggestions and insights for educators and researchers on using AIGC-supported programming courses. Based on this, the following research questions are proposed:

1. Does LLM-based collaborative programming improve students' computational thinking more than conventional collaborative programming?
2. Does LLM-based collaborative programming improve students' self-efficacy and reduce cognitive load more than conventional collaborative programming?

## Literature

**Education and computational thinking.** Computational thinking, which refers to solving problems, designing systems, and understanding human behavior through the use of basic concepts in computer science, is a fundamental skill necessary for every human being (Nouri et al. 2020). Since Wing (Wing 2006) introduced the concept of computational thinking, her argument has provided a new perspective on the relationship between humans and computers has triggered a wave of research on computational thinking. Several studies have emphasized the positive correlation between students' programming skills and their improved computational thinking (Nouri et al. 2020; Rich et al. 2017). It was explained that the development of students' programming skills and computational thinking skills are positively correlated (Belmar 2022), and that the improvement of students' programming skills will correspondingly improve their computational thinking (Oluk and Korkmaz 2016). The correlation between the development of computational thinking and programming education emphasizes the importance of programming education for improving students' computational thinking skills (Belmar 2022; Nouri et al. 2020; Tikva and Tambouris 2021).

In order to cultivate new youth who can adapt to the rapidly evolving social era, governments and leading educational organizations are giving importance to coding education and computational thinking (Hsu et al. 2019; Williamson 2016). As early as 2014, the European School Network published a survey report stating that 13 out of 51 countries participating in the survey have introduced or will soon introduce coding in K-9 education (Mannila et al. 2014).

Collaborative programming was developed based on collaborative learning, which, according to cognitivism and constructivism, refers to the active and collaborative construction of knowledge by each individual in a learning group based on previously learned conceptual knowledge through individual learning experiences and social interactions with other learners in the team (Kalaian and Kasim 2015). Collaborative programming has long been considered an effective way to improve the effectiveness of programming instruction (Johnson and Johnson 1987), which can help team members work together to overcome their programming shortcomings (Edmondson 1999). Especially for novice programmers, collaborative programming can be a good way to help learners recognize knowledge about computer programming and help them build computational thinking and computer programming skills (Zheng et al. 2022).

However, in the conventional collaborative learning process, learners tend to believe that they can achieve common learning goals when working with other group members (Kalaian and Kasim 2015). It is worth exploring this perspective further, as research has shown that the composition of a team and the competencies of its members are related to the overall performance of the team (Cohen and Bailey 1997). The diversity of skills, experience, and knowledge of team members may affect the effectiveness of teamwork, and the collaborative programming process may be hindered by common shortcomings and difficulties in the programming skills of group members (Strom and Strom 2002; Xinogalos et al. 2017).

**Self-efficacy in collaborative programming.** The conceptualization of self-efficacy is one of the most prominent aspects of Bandura's social-cognitive/learning theory of personality development (Bandura 1978), which suggests that students with an enhanced sense of efficacy will have higher expectations of themselves and demonstrate greater strategic flexibility in finding solutions than students with the same cognitive abilities. It is crucial for students to be able to think and solve problems independently in programming tasks, and individuals with high levels of self-efficacy are more likely to seek effective solutions to difficulties rather than avoid problems (Bandura 1978). According to Bandura's social cognitive theory, self-efficacy can be improved through observation, imitation, and feedback (Bandura 1978). During collaborative programming, students observe each other and share problem-solving strategies, which helps to improve their self-efficacy (Yilmaz and Yilmaz 2023b; Zheng and Zheng 2021). Yuliyanto in his study pointed out that learners' cognitive load shows a direct correlation with self-efficacy (Yuliyanto et al. 2019). The lower the cognitive load of the students, the higher the self-efficacy and the more it helps the students to learn (Vasile et al. 2011). In complex programming tasks, the complexity of tasks and problems can be shared among team members, while team members can share knowledge and resources, and these practices reduce the cognitive load of complex tasks (Zhang et al. 2016). In collaborative programming, the presence of other members provides a double-checking mechanism that helps to confirm individual solutions, and such a feedback loop reduces the cognitive load of the students during their initial exposure to the programming syntax (Chang and Wongwatkit 2023).

Students in elementary to middle school need to face the transition from graphical to coded programming, and the shift from concrete to abstract causes a sharp increase in students' cognitive load, and learners' self-efficacy feels a huge shock (Louca and Zacharia 2008). Kalaian and Kasim explored the effects of concrete and abstract teaching styles on learners' self-efficacy in his study (Kalaian and Kasim 2015), the results of this study showed that figurative teaching styles are more likely to increase learners' self-efficacy (Yuliyanto et al. 2019), but according to Bloom's Cognitive Goal Classification System (Bloom et al. 1956), abstract teaching styles are more likely to support students' development of higher-order thinking skills, and therefore how to maintain students' self-efficacy during this transition is a very important issue.

### Large language models (LLMs) and its potential in education.

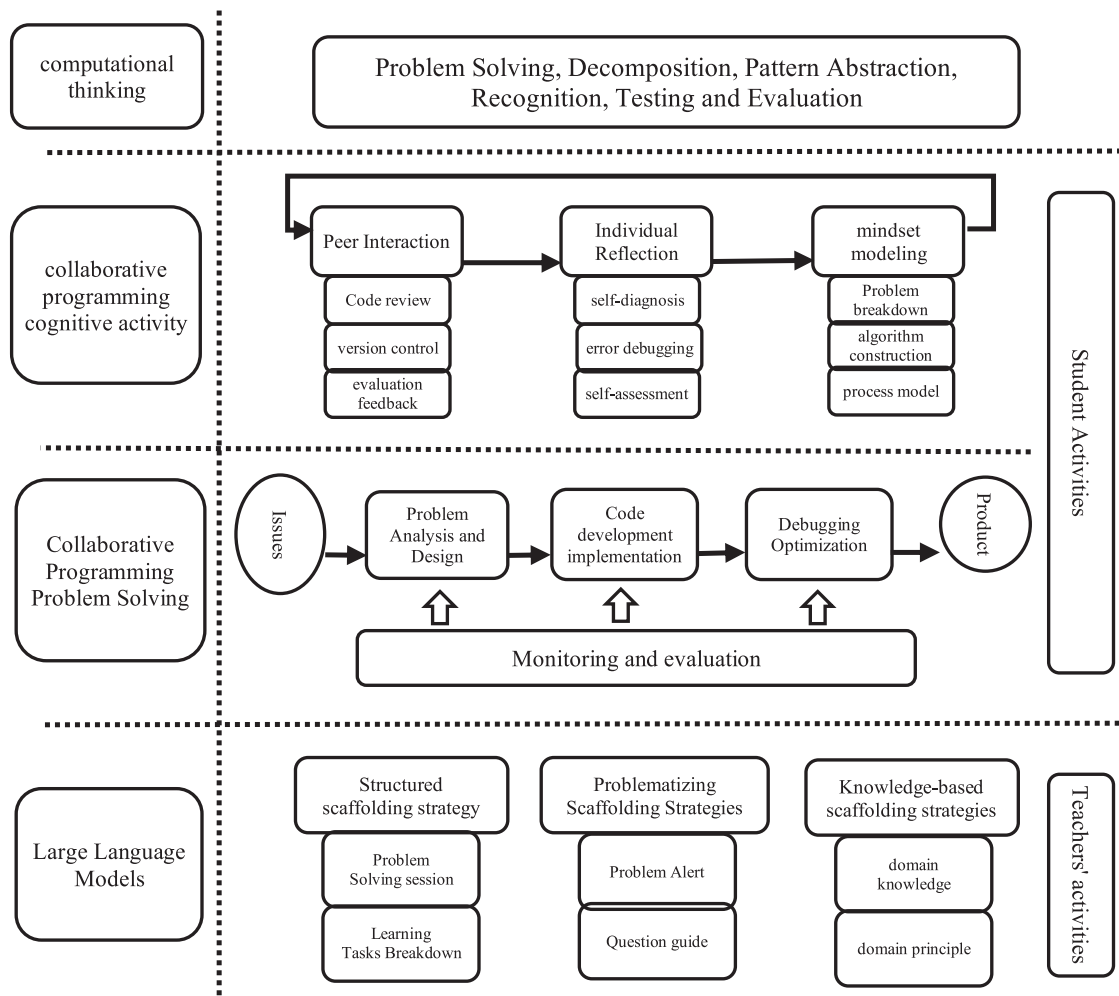
To effectively address the challenges posed by socioeconomic changes and the growing demand for AI in the digital economy, AIGC has emerged. AIGC refers to content created using advanced generative AI (GAI) techniques, as opposed to manual creation (Wu et al. 2023). Advances in LLMs have significantly improved the performance of AIGC, giving it new potentials and opportunities, and injecting a lot of convenience into our daily lives (Wu et al. 2023). LLM is an Artificial Intelligence (AI) tool based on multilayer recurrent neural networks, which are trained on large amounts of data to generate human-like texts (Alberts et al. 2023). Some representative LLMs include Pathways Language Model, Generalist Language Model, Generative pretrained Transformer (GPT) (Valmeekam et al. 2022). LLMs have unconsciously influenced every aspect of people's lives, especially in the field of education (Alberts et al. 2023). Existing and emerging technologies will help teachers to do their educational teaching better and more effectively (Bryant et al. 2020). AI technology was utilized to design educational robots, use AI technology in the classroom, and use educational robots to help

students teach robotics programming (O'Hara et al. 2015). In programming education, LLMs can play the role of a programmer by writing structured query language queries and executing them, and can even write Python, Verilog, and C++ codes briefly according to the needs, simulate training networks, and simulate Linux systems and other operations (Yilmaz and Yilmaz 2023a). In addition, LLMs can also provide personalized tutoring to students, provide personalized guidance and feedback to students based on their individual learning needs and progress (Kasneji et al. 2023), reduce students' participation in automated and repetitive tasks, thus reducing students' cognitive load so that students can focus on problem solving and algorithm design (Hutson and Plate 2023), and to a large extent solve the difficulties brought by collaborative programming, change the original collaborative programming learning structure from human plus human model to human plus human combine AI model. In Arora's research, the use of Master of Law in higher education programming has been underestimated. A total of 411 students participated in this study, and the results showed that students completed various programming tasks using LLM, including code generation and debugging, concept queries, and example creation. Students generally believe that a Master's degree in Law can help improve their programming and learning abilities (Arora et al. 2024). Yilmaz and Karaoglan Yilmaz conducted an 8-week-long experiment with 41 higher education students who were asked to use LLM in problem solving, and at the end of the study, the students reported that using LLM in their programming learning allowed them to get most of the correct answers to their questions quickly (Yilmaz and Yilmaz 2023a), which could improve their thinking skills, facilitate their debugging, reduce their cognitive load and enhance their self-efficacy. In the study by Ouazaki, LLM was used to simplify code generation and debugging, enhancing students' interactive learning methods. The research results showed that using ChatGPT as a learning assistant can improve students' learning outcomes (Ouazaki et al. 2023). In addition, Wang tested the ability of LLMs to solve complex problems in his research, and the results showed that customized prompting strategies can significantly improve LLMs' problem-solving ability. This study systematically classified and tested different strategies, providing a comprehensive framework for educators and students, and reminding us to set certain strategies when using LLMs as problem-solving tools to more effectively improve the educational outcomes of computer programming teaching (Wang et al. 2024). Kulangara and Kiran integrated multiple LLMs into a specific platform to enhance programming education. They researched and customized an introductory programming learning platform, where students can receive personalized learning experiences with the feedback and assistance of LLMs. Despite the profound impact of the GPT-4 on education, there is still a lack of existing research on empirical studies conducted in this area, especially among students at the primary level of education, whereas programming is considered difficult by many (Becker et al. 2023), especially among students at the k-12 education level.

Therefore, it is important to consider that the integration of LLMs technology into the teaching and learning environment offers exciting opportunities to transform the learning and educational process. This study investigates the role of LLMs supported collaborative programming learning methodology on the development of learners' computational thinking skills in a programming education classroom for middle school students.

### Design of LLMs-based collaborative programming framework

This study aims to form a framework for collaborative programming based on LLMs with the basic goal of developing students'



**Fig. 1** LLM-based collaborative programming framework diagram.

computational thinking during programming teaching and learning, supported by the increasingly mature LLMs tool for collaborative programming. This framework is based on social constructivist theory because social constructivist theory emphasizes the co-construction of knowledge, the social nature of learning, and the importance of cooperative learning, which are closely related to the goals and principles of collaborative programming (Looker 2021). Currently, more and more researchers are focusing on the application of GAI in collaborative learning models (Tan et al. 2023). Meanwhile, combining the operational definition of computational thinking skills for K-12 education in primary and middle schools as the basis of teaching design, we propose a teaching process based on the main line of “problem analysis and design—code development and implementation—debugging and optimization—monitoring and evaluation” (Fig. 1).

In this study, the original collaborative learning strategy was modified to better fit the offline programming teaching process, and the function of LLMs was integrated, in which the LLM serves as a scaffolding strategy (structured scaffolding, problematized scaffolding, and knowledge-based scaffolding) for learning and plays a role in facilitating the collaborative programming problem-solving session and students’ cognitive activities, and these scaffolds reduce the cognitive load of students’ learning process while increasing their self-efficacy, which in turn improves computational thinking, in particular:

LLM as a structured scaffolding strategy optimizes the learning process in three main ways: First, by decomposing the problem-

solving process into four core aspects of collaborative programming and providing a standard template to clarify the direction of learning; second, by embedding peer interaction, individual reflection, and thinking tools in this process and using LLM to visualize the cognitive activities and enhance the continuity of learning; and finally, by decomposing the complex tasks into operational subtasks, LLM reduces the cognitive load on learners in organizing the learning process and increases students’ programming self-efficacy. LLM reduces the complexity of the learning process and the number of tasks through technological means, allowing learners to focus more on key problem-solving activities, thus improving the efficiency and effectiveness of collaborative programming.

LLM as a pedagogical strategy for problem scaffolding is used throughout the problem solving and cognitive activity process and plays a key role in clarifying goals, facilitating higher-level cognitive activity, and continuous improvement. First, in the problem solving and design process, LLM-generated problem scaffolding facilitates cognitive and experiential connections between learners and their peers. In code development and implementation, problem guidance can facilitate the negotiation and construction of higher-level meaning. Further, in the debugging and optimization loop, teachers and LLMs work together to help learners identify and improve the project’s shortcomings through problem guidance. The monitoring and evaluation loop are involved in each session, and the LLM’s question prompts are used to guide learners to reflect on the

learning process and milestones, thereby generating new questions that further facilitate meaning construction.

LLM serves as a knowledge scaffolding strategy in the problem solving and design phase, where LLM generates targeted instructional materials to facilitate more accurate problem interpretation and analysis based on learners' knowledge blind spots and weaknesses. Second, in the code development and implementation phase, LLM provides working examples of domain principles and problem solutions to help learners establish procedural rules and store them in long-term memory through analogies and example explanations. In the debugging and optimization phase, LLM can diagnose learners' knowledge deficits and provide targeted reinforcement materials or new work samples to optimize their cognitive performance. In the monitoring and evaluation phase, scaffolding strategies focus on facilitating learners' cognitive and metacognitive knowledge and strategies to stimulate new cognitive conflicts and reflection. These knowledge-based scaffolds are usually provided in the form of learning resources and can be used in combination with problematization and modeling scaffolding strategies.

Materials and methods

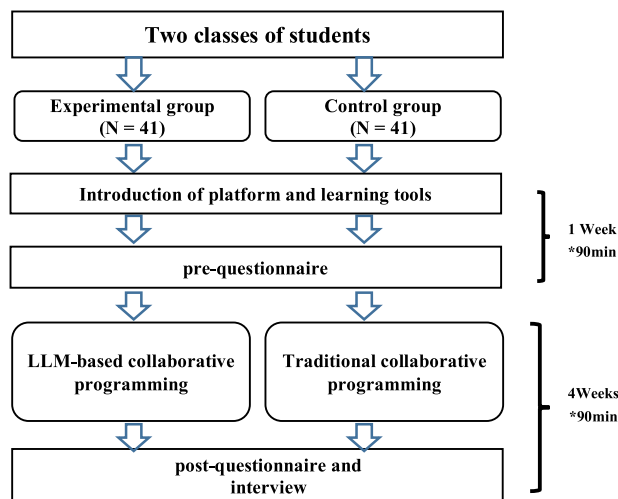
**Participants.** The participants are students from sixth to seventh grade at a Chinese high school that implements a nine-year integrated education system. Therefore, the socio-economic level and technology acquired by students are basically similar, with a total of 85 students participating. After excluding invalid sample data from students who did not participate in the pre-test and post-test, a total of 82 valid samples were obtained, including 82 sixth to seventh grade students (41 boys and 41 girls) (see Table 1). Students participated in the experiment with written informed consent from their parents or guardians. The development of the research plan has been approved and participated in by the principal and multiple teachers in the grade. The students participating in the experiment had all been exposed to Scratch graphic programming in elementary school and had not been exposed to C++ language. Before the formal teaching, a graphical programming skill test was conducted on the students in the experimental group and the control group, with scores of  $(15.900 \pm 2.468, 16.225 \pm 2.247)$ , respectively. A *t*-test was used to prove that there was no significant difference in programming skills ( $t = -0.466, p = 0.642$ ).

**Procedure.** Programming classes are taught by an instructor with over ten years of teaching experience to ensure consistency in content, pace, and instructor level. The language taught in the programming classroom is specified as C++, and Dev C++ programs are used for compilation. Students will work in small groups (4 students per group) to answer questions in the programming lab.

This study used a randomized controlled trial to evaluate the effects of incorporating LLMs into collaborative programming instruction. Participants came from multiple classes at the same grade level and were randomly assigned to either the control or LLMs group using a simple randomization procedure. Over a five-week period, all participants participated in 12 programming sessions (90 min each). The experimental group learned in LLMs-based collaborative programming, while the control group followed conventional collaborative programming methods. Prior to the start of the study, a pretest of computational thinking and a pretest of self-efficacy were administered to all students, and the teaching process in the experimental group involved the teacher instructing the students in the use of LLMs in addition to teaching about programming. After the four-week intervention, the students were given the computational thinking post-test, the

Table 1 Students information.

Gender	Group	Age						Total	
		10			11			12	
		N	Per.		N	Per.		N	Per.
Male	Experimental	20	24.4%	Control	19	23.2%		10	12.2%
Female		21	25.6%		22	26.8%		5	6.1%
Total		41	50.0%		41	50.0%		15	18.3%
								82	100%



**Fig. 2** Experimental procedure.

self-efficacy post-test, cognitive load test and interview, the flow chart of the experiment is shown in Fig. 2.

**Measuring tools.** The experiment was conducted using the researcher's self-administered computational thinking test papers (A and B) as the measurement tools for the pre- and post-tests, and all the questions in the test papers were selected from the Bebras International Computational Thinking Challenge Test Book (Hubwieser & Mühling, 2015), so the questions were well differentiated.

The self-efficacy pre- and post-test questions were adapted from Wang and Hwang (Wang and Hwang 2012) and consisted of eight items, such as "I believe that I can understand the most difficult part of the class". This questionnaire used a 5-point Likert scale ranging from 1 = not at all to 5 = completely. The Cronbach's alpha for this questionnaire in the original study was 0.92.

In order to further assess the effect of LLMs collaborative programming on self-efficacy, we measured the cognitive load of the students' learning process, and the questionnaire used the scale revised by Hwang (Hwang et al. 2013). The scale consisted of eight questions. A six-point Likert scale was used, with 1 representing complete noncompliance and 6 representing complete compliance. The Cronbach's alpha values for the two dimensions were 0.86 and 0.85, respectively.

In the post-test phase of this study, 10 students from each of the experimental and control groups will be randomly selected to conduct semi-structured interviews. These interviews will be based on Liang and Hwang (Liang and Hwang 2023) interview template with appropriate modifications and will include a total of seven questions. Each participant's interview is expected to last between 10 and 15 min. Through this series of semi-structured interviews, we aim to explore in depth students' perceptions and views of conventional collaborative programming versus LLM-based collaborative programming teaching methods. This will help us conduct a qualitative study to further understand the potential impact and value of LLMs in education.

**Experiment platform.** SpeechGPT, an intelligent dialogue platform published on GitHub, enables learners to interact fluently with advanced AI models for a series of intelligent dialogues. However, the existing features of the SpeechGPT platform are limited and difficult to meet the needs of collaborative programming education. To realize the incorporation of LLM technology into collaborative programming education, the research

team carries out the secondary development of the SpeechGPT platform. The secondary development is mainly designed in two key areas: prompt design and multimodal input. The first is to optimize the prompt design to improve the effectiveness of interaction between learners and LLMs, and to guide learners to ask more targeted questions through the relevant prompts to improve the effectiveness and accuracy of knowledge sharing. The second is multimodal information input, focusing on the speech input module in the interface to provide students with more convenient input and output feedback to meet the cognitive needs of different students.

Online Code Assessment Platforms are software solutions that automate the evaluation and grading of programming assignments or contest entries. These platforms are able to support multiple programming languages such as Python, C, C++, Java, etc. and can automatically execute predefined test cases to evaluate the correctness, efficiency, and robustness of the code. In this study, the programming assignments assigned by the teacher are published on this platform, and the teacher can view the students' work in real time to adjust the teaching strategy. For detailed instructions on the operation of prompt language, please refer to the appendix (modular prompts for GPT-4) Fig. 3.

This study employed the GPT-4-0314 model and instituted specific API configurations, including:

OpenAI API Address: chat.openai.com

OpenAI model: GPT-4-0314

System Role: You are an experienced C++ programming language expert, and now you need to become a learning partner for a team of four students. You can provide ideas for their problems, help them debug and optimize code, but you cannot directly provide them with complete problem code. Each problem has four programs: "Problem Analysis and Design"—"Code Development and Implementation"—"Debugging and Optimization"—"Monitoring and Evaluation"

Degree of divergent thinking: 0.2

Number of consecutive conversations: 300.

**Data analysis.** Analysis of covariance (ANCOVA) in SPSS26 tool was used to compare the differences in computational thinking awareness and self-efficacy between the two groups of students after the intervention. The auxiliary measurements included qualitative and quantitative parts, for the quantitative measurement of cognitive load, the independent samples *t*-test was used to compare the cognitive load of the two groups of students in the learning process, the qualitative data included the qualitative interviews with the students after the experiment, which were recorded by the KDDI audio recorder and then transcribed into text for coding and analysis.

## Results

**Computational thinking awareness.** Before conducting a one-way ANCOVA on computational thinking, the study examined linear relationships, normality, homogeneity of variance, and parallelism. The correlation coefficients were relatively high, indicating the presence of linear relationships (Pearson's  $r > 0.3$ ;  $p < 0.02$ ). The residuals of the dependent variable followed a normal distribution ( $p = 0.13 > 0.1$ ). The results of the Levine test indicated that the assumption of homogeneity of variances was met ( $F = 0.05$ ,  $p = 0.83 > 0.05$ ). The interaction term was not statistically significant, satisfying the parallelism test ( $F = 1.66$ ,  $p = 0.20$ ), indicating that the analysis of ANCOVA can be used.

After excluding the effect of the pre-test to the level of computational thinking, the scores of the experimental and control groups showed a significant difference ( $F = 4.39$ ,  $p < 0.05$ ) between the scores of the experimental group ( $M = 81.35$ ,

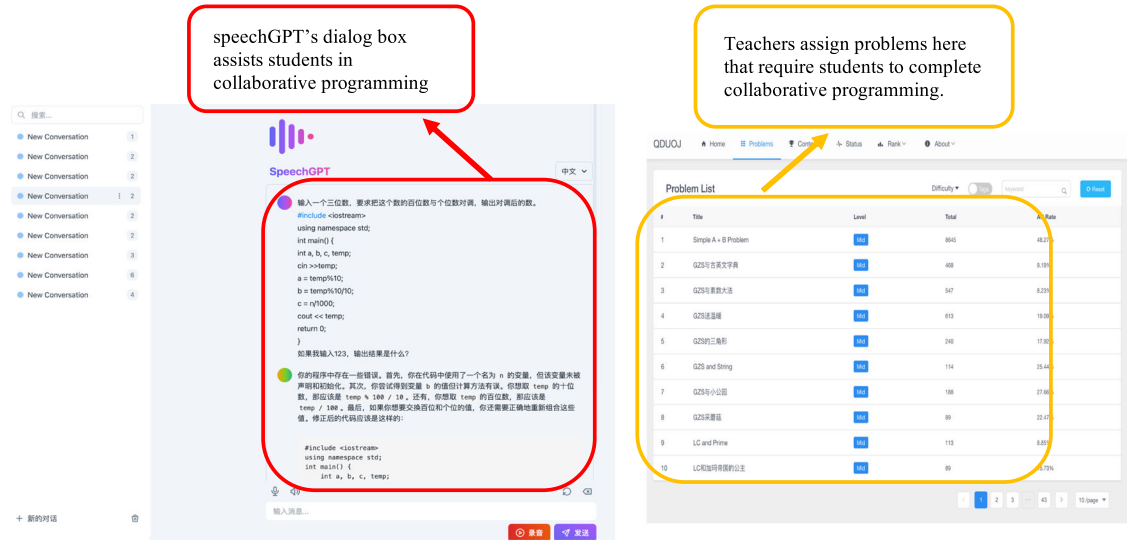


Fig. 3 Experiment platform.

Table 2 The ANCOVA result of computational thinking awareness for the two groups.							
Variables	Groups	N	Mean	SD	Adjusted mean	F	$\eta^2$
Computational thinking	Experimental group	41	81.35	11.51	81.31	4.39*	0.04
	Control group	41	73.08	11.86	74.12		
* $p < 0.05$ .							

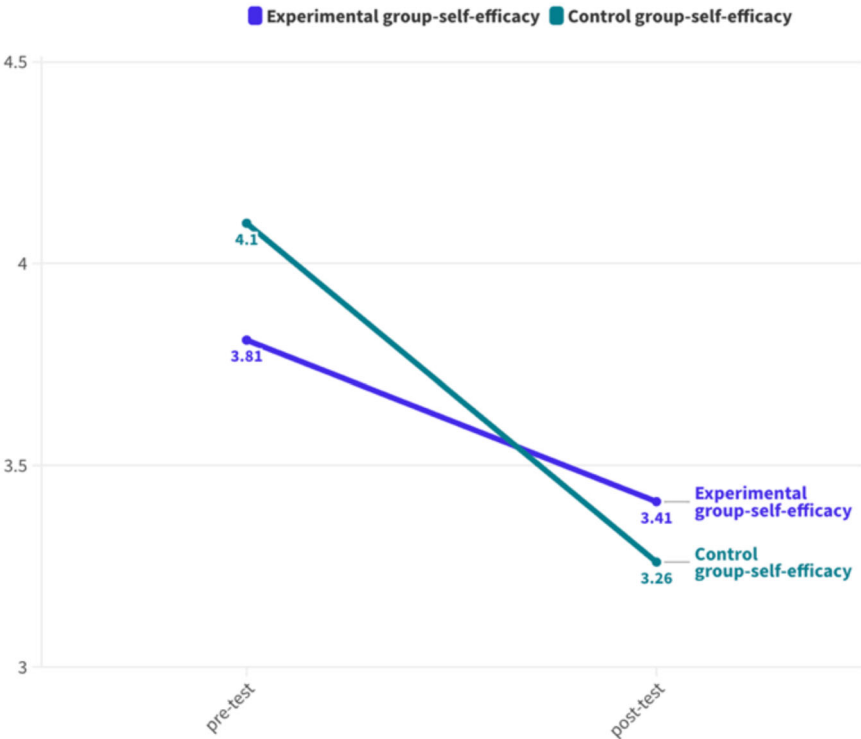
Table 3 The Mann-Whitney U test result of Self-efficacy post-test for the two groups.							
Variables	Groups	N	Mean	SD	Adjusted mean	U	$\rho$
Self-efficacy	Experimental group	41	3.41	0.83	3.48	756	−0.101
	Control group	41	3.26	0.92	3.19		

SD = 11.51) and the scores of the control group ( $M = 73.08$ ,  $SD = 11.86$ ).  $\eta^2$  effect size was 0.04, which indicates a small effect size (Cohen, 2013) (see Table 2). This suggests that there is a statistically significant difference in the post test of computational thinking between the experimental group and the control group. The smaller effect size may be due to students just transitioning from graphical programming, resulting in insufficient abstract ability and limited improvement in computational thinking when facing programming languages with steep learning curves like C++. Therefore, in the real teaching environment, teachers should consider the learning difficulty of C++ when designing teaching strategies, avoid excessive reliance on a single technical tool or intervention strategy, and adopt more comprehensive teaching methods, combined with technical support, peer discussion, and teacher guidance, to more effectively promote students' learning and ability improvement.

**Self-efficacy.** Due to the non-normal distribution of pre-test and post-test data in the experimental group for self-efficacy ( $p = 0.008$ ,  $0.004 < 0.05$ ) and the non-normal distribution of pre-test and post-test data in the control group ( $p < 0.001$ ), the Mann-Whitney  $U$  test was considered. At the pre-test level, there was no significant difference in scores between the experimental group ( $M = 3.81$ ,  $SD = 1.00$ ) and the control group ( $M = 4.10$ ,  $SD = 0.83$ ) ( $U = 701$ ,  $p = 0.19 > 0.05$ ). At the post-test level, there was no significant difference in scores between the experimental group ( $M = 3.41$ ,  $SD = 0.83$ ) and the control group

( $M = 3.26$ ,  $SD = 0.92$ ) ( $U = 756$ ,  $p = 0.43 > 0.05$ ) (see Table 3). Although the average post-test score of the experimental group was slightly higher than that of the control group, this difference was not statistically significant. Additionally, a comparison of pre-test and post-test measures of self-efficacy showed that both the experimental group and the control group exhibited a certain degree of decline, with the experimental group showing a relatively smaller decline and the control group showing a larger decline (see Fig. 4).

Based on the data in Table 4, we compared the performance of the two groups of students on three dimensions: cognitive load, and its sub-dimensions mental load and mental effort. Before conducting a Independent samples T-test analysis of student cognitive load, the study examined variable followed a normal distribution ( $p > 0.05$ ). Firstly for cognitive load, the experimental group had a mean of 3.96 with a standard deviation of 0.765, while the control group had a mean of 4.39 with a standard deviation of 0.887. the  $t$ -test was statistically significant at  $-2.20$  ( $p < 0.05$ ), and the effect size,  $d$ , was  $-0.518$ , which presents a moderate effect size. This means that the experimental group performed significantly lower than the control group in the cognitive load dimension and this difference was statistically significant. For the dimension of mental load, the mean of the experimental group was 3.91 with a standard deviation of 0.713, while the mean of the control group was 4.38 with a standard deviation of 0.844. The result of the  $t$ -test was  $-2.51$ , which is statistically significant ( $p < 0.05$ ), and the effect size  $d$  was  $-0.592$ ,



**Fig. 4** Pre- and post-test means of self-efficacy.

Table 4 Independent samples T-test analysis of student cognitive load.						
Variables	Groups	N	Mean	SD	t	d
Cognitive load	Experimental group	41	3.96	0.765	-2.20*	-0.518
	Control group	41	4.39	0.887		
Mental load	Experimental group	41	3.91	0.713	-2.51*	0.0592
	Control group	41	4.38	0.844		
Mental effort	Experimental group	41	4.01	0.927	-1.70	
	Control group	41	4.41	1.037		

\*p < 0.05.

which presents a medium effect size. This result likewise indicated that the experimental group also had lower scores on the mental load dimension than the control group, and the difference was statistically significant. For the mental effort dimension, the experimental group had a mean of 4.01 with a standard deviation of 0.927, while the control group had a mean of 4.41 with a standard deviation of 1.037. The result of the *t*-test was  $-1.70$ , which did not meet the criteria for statistical significance. This means that there is no significant difference between the two groups in terms of mental effort.

**Interview.** In order to further understand the role of LLMs in assisting collaborative programming, we randomly selected 10 students from the experimental group and the control group respectively for interviews. After the interviews, we numbered these 20 students, with E1–E10 representing the experimental group and 1–10 representing the control group, and coded the interviews content as follows. The first step is data transcription and initial reading, transcribing the interviews recording into text

and reading each interviews record one by one to obtain a comprehensive understanding of the data. The second step is open coding, where we divide each paragraph, sentence, or answer in the interviews into small units and label them with code. For example, when discussing “increasing interest”, the students in the experimental group mentioned that “LLMs as a programming partner has enhanced my interest”, which is encoded as “LLMs increases programming interest”, laying the foundation for subsequent topic classification. The third step is axis encoding, which aggregates and classifies the labels extracted during the open encoding stage to form a more macroscopic theme. For example, feedback from students about “increased interest” is categorized as “advantages”, while feedback about “poor network” is categorized as “disadvantages”. In addition to interviews data, we can also combine the specific classroom situation of students to observe whether they show higher participation or more tendencies when using LLMs, in order to verify the theme of “increasing interest” in the interviews data. In addition, during the coding process, we invited other researchers to participate in data analysis to ensure the objectivity of the analysis process. The research team members independently encoded the data and then combined them for collective discussion, in order to reduce the influence of personal biases of researchers and improve the accuracy of analysis.

As shown in Table 5, within the “advantages” theme, the experimental group responded more positively than the control group in terms of “increased interest”, “more thoughtful”, and “easier”. Specifically, regarding “increased interest”, most students in the experimental group mentioned that integrating LLMs as a collaborative partner into programming significantly boosted their interest in programming, while half of the control group students believed that collaborative programming alone could increase their programming interest. In terms of “more thoughtful”, the collaborative programming with LLMs provided different hints and solutions, encouraging experimental group students to think more comprehensively, while the control group

**Table 5 Frequency analysis table of interviews results.**

Theme	Theme code	The number of occurrences	
		Experimental group	Control group
Advantages	Boosting interest in programming	8	5
	Enhance self-confidence	6	5
	Easier	7	4
Drawbacks	More thoughtful	6	3
	Misunderstand the meaning	5	0
	Content is difficult to understand	4	2
	Poor network	3	0
Student Gains	Problem-solving skills	6	5
	Critical thinking	3	3
	Communication and collaboration skills	4	2

students had to rely solely on their existing knowledge base to solve problems. Regarding “easier”, LLM-assisted collaborative programming helped reduce their cognitive load on complex concepts and simplified the difficulties in the programming process, whereas the control group students reported spending a considerable amount of time consulting external materials when faced with challenges. Additionally, more students in the experimental group reported that LLMs enabled them to solve problems faster during programming and made the process of working together more enjoyable. This suggests that LLMs assisted collaborative programming can improve the quality and efficiency of student code, and reduce their cognitive load in programming. On the other hand, this also proves that integrating LLMs into collaborative programming can significantly reduce students’ cognitive load in programming groups. Within the “student gains” theme, the experimental group performed better in “problem-solving skills” and “communication and collaboration skills”, indicating that LLMs integrated into collaborative programming provide students with a better environment for cooperation and communication. The vast knowledge base of LLMs helped students improve their problem-solving abilities, demonstrating the effectiveness of incorporating LLMs into collaborative programming instruction.

## Discussion

This study proposes a model of collaborative programming teaching strategies based on LLMs and determines the effectiveness of collaborative programming supported by LLMs based on an experimental design that utilizes the generative artificial intelligence support to the experimental group, and the control group receives conventional collaborative programming education. In this study, the level of computational thinking, programming self-efficacy and cognitive load of students in the experimental and control groups were compared.

Addressing the first research question, the results of the study showed that the collaborative programming group based on LLMs got more improvement than the conventional collaborative programming group in terms of computational thinking level. Observations from the application process understood that the application of LLMs provided immediate feedback and support in solving programming problems and conceptual understanding, which accelerated the students’ knowledge construction process. The interviews results showed that more students in the experimental group said that LLMs enabled them to solve problems faster in the programming process and made the process of working together more enjoyable. In conventional collaborative programming environments, students may need to spend more time and effort searching for relevant information or waiting for feedback from teachers and peers. However, in collaborative

programming environments based on LLMs, students can quickly access solutions, code samples, and conceptual explanations for specific problems or programming tasks. More importantly, LLMs are able to generate targeted and contextually relevant feedback based on queries or problems entered by students. This not only reduces the time cost of information retrieval, but also improves the accuracy and usability of the information.

The results of the current study on computational thinking are in line with Yilmaz and Karaoglan Yilmaz (Yilmaz and Yilmaz 2023b), in which it was confirmed that the use of generative artificial intelligence in programming education leads to better levels of computational thinking. Researchers proposed the inevitability of future human-computer collaborative learning, where the role of technology in human learning changes from a mere tool to an environment, a partner, and ultimately may become one with humans (Fui-Hoon Nah et al. 2023), and according to the theory of social constructivism, LLMs can act as a “mediator” to According to the social constructivist theory, LLMs can act as a “mediator” to facilitate more effective social interactions and discussions among students, and cooperation and interaction can promote the development of their computational thinking (Sharma et al. 2019), thus indirectly improving their computational thinking skills. Interviews at the end of the experiment showed that students in the lab group allocated more time to algorithmic thinking and discussion for problem solving than the control group. Thus, providing students with AI training and using AI-supported tools in collaborative learning is effective in improving their computational thinking skills.

In response to the second research question, in terms of self-efficacy in programming, the results of the study indicated that there was no significant difference in self-efficacy between the two groups of students, and that both the experimental and control groups showed some degree of decline in programming self-efficacy compared to the pre-test. The randomized interviews study revealed that many of the students were being exposed to code programming for the first time, that previous programming classes had used scratch graphical programming, and that the shift from graphical to textual programming was a major cognitive leap. During this process, students need to adapt to different programming paradigms and language structures, and the complexity of textual programming may still have some impact on self-efficacy, leading to a decrease in students’ self-efficacy (Chen et al. 2019). Nevertheless, in terms of the degree of self-efficacy decline, the decline in collaborative programming based on LLMs was significantly smaller than that of conventional collaborative groups without LLMs (see Fig. 4). It can be seen that LLMs increased self-efficacy in collaborative programming to a certain extent, thanks to the strategy guidance, code review and debugging feedback provided by LLMs, which made the students

less fearful of programming in code, and some researchers have pointed out that intelligent learning environments built by AI have a positive effect on learners' conceptual transformation (Chen et al. 2023), which is very meaningful for students at the stage of programming paradigm shift. In addition, from a social constructivist perspective, LLMs can facilitate group discussion and reflection by providing students with a discussion structure and real-time feedback during discussions (Kasneji et al. 2023), which can help to improve students' engagement and self-efficacy, and a qualitative study on AIGC reached similar conclusions (Yilmaz and Yilmaz 2023a). It can be said that these studies to some extent support the positive impact of LLMs on programming self-efficacy, and the results of random interviews also support this conclusion. From the interviews results, most students believe that using LLMs improves their programming self-efficacy, which proves from another perspective that integrating collaborative programming into LLMs can increase students' self-efficacy in programming groups. This can be further verified through future research in a wider population and compared with the group effects of collaboration.

The results of cognitive load in the current study also support the conclusion of self-efficacy to some extent, as students' self-efficacy is highly correlated with their cognitive load. The experimental group was provided with personalized instruction in LLMs technology, and this support may have reduced their cognitive load and psychological load, which is consistent with the findings of Herm who mentioned that interpretable AI can be effective in reducing students' cognitive load (Herm 2023), whereas the control group may not have been provided with the same level of support in adapting to the code-based programming approach, which resulted in an increase in cognitive load and psycho-logical load. On the other hand, based on the interviews results and research conclusions, the experimental group had the scaffolding strategy provided by the LLMs, where students could ask questions and collaborate to solve them within the code framework. In this process, learners were not limited to a single task and could be more actively engaged and creative, resulting in an untargeted effect of cognitive load. This may be the reason why the experimental group was not only able to achieve better computational thinking but also lower cognitive load (Antonenko and Niederhauser 2010)."

From the perspective of design limitations, the decrease in self-efficacy observed in the research results is a finding worthy of further exploration. This discovery requires a critical perspective to analyze the limitations of research design. This study uses C++ language as the main learning content for programming classes, which is too obscure and difficult to understand. With the participation of LLMs, students may feel that the process of problem-solving becomes easier, but in the process of coding alone, students' self-efficacy may not be as good as in the era of graphic programming. This indicates that in the teaching process, teachers can gradually transition to the C++ programming language from graphical programming or simplified programming environments before formally teaching the language. This can help students build confidence from the beginning. In addition, teachers can clearly inform students of the boundary between independent thinking and LLMs discussing problem-solving, further plan the functions of LLMs, and enable students to utilize LLMs through gradual guidance, discussion, and reflection.

## Conclusions

This study provides promising preliminary evidence supporting the effectiveness of strategies that integrate LLMs into collaborative programming. Compared to conventional collaborative

programming strategies, the study found that collaborative programming based on LLMs substantially improved students' computational thinking, LLMs acted as learning scaffolds for the students' learning process, and the students were able to participate more effectively in collaborative programming to a certain extent by getting rid of the boring literacy process of conventional programming, although the students' self-efficacy did not show a statistically significant difference. However, we obtained positive findings about LLMs from trends in self-efficacy, student interviews, and cognitive load data. These findings emphasize the potential for targeted integration of AI technologies into established educational paradigms to effectively enhance classroom instruction, as well as areas for continued innovation and research. As educational technology continues to evolve, anchoring technology applications in well-established learning theories and empirical best practices remains imperative. This study exemplifies this integration by proposing a collaborative programming strategy model based on LLMs. Further research based on these principles can help guide the judicious adoption of AI to unlock its transformative possibilities for the future of education.

Based on the research process and results, the study makes some recommendations for researchers and educators. First of all, it should be taken into account that LLMs are a text-based chatbot that can give instant answers to questions and that the tool gives mostly correct and logical answers to text-based questions. In this case, what teachers should do is not to assign homework/problems, but to develop the ability of students to ask questions to LLMs to get answers, and in the future assignments will no longer be given to get a single solution, but to enable students to think in different ways. Secondly, considering that many studies mention the dependency problem of LLMs, teachers should pay particular attention to the development of students' critical thinking in the process of using them, and the introduction of metacognitive tools to promote students' reflection is a better way to do so, and the importance of metacognitive scaffolding in collaborative programming had been proved (Zhan et al. 2022). Finally, the collaborative programming model based on LLMs is well suited to project-based learning, where future teaching sessions are student-driven through group-based self-directed inquiry.

**Limitations and further research.** This study provides valuable insights into incorporating LLMs into collaborative programming instruction, but there are several limitations. First, the intervention period was only four weeks, and students' self-efficacy showed a decreasing trend during this period, which limits our understanding of the long-term impact of LLMs on collaborative programming. Additionally, unequal access to technology could have introduced potential biases, as students from different socioeconomic backgrounds may have had varying levels of access to necessary technological resources, such as stable internet, devices, and technical support. Second, the study was limited to students at the K-12 level, which is a critical period when students are shifting from figurative to abstract thinking, limiting the external validity of the findings. Finally, this study only explored the effects of LLMs in collaborative programming and provided an integration of LLMs with other programming instructional models, which did not allow for testing the generalizability of the use of LLMs in programming instruction.

To address these limitations, future research should expand both horizontally and longitudinally. Horizontally, the scope of research should be broadened to include students from a wider range of backgrounds and educational levels to assess whether LLMs can benefit all students equitably. Longitudinal studies are

also needed to observe the long-term effects of LLM-based collaborative programming instruction on students' computational thinking, programming skills, self-efficacy, and cognitive load. Moreover, future research should explore strategies for mitigating disparities in technological access, ensuring that all students have an equal opportunity to benefit from LLM-assisted learning. This series of investigations will help build a more comprehensive understanding of the potential impact and long-term effects of LLM technology in programming education.

## Data availability

Due to the reason why the data is not publicly available, the datasets generated and/or analyzed during the current research period are not publicly available, but can be obtained from the corresponding authors upon reasonable request. Due to poor data storage, the cognitive load data in the experimental results of this study cannot be provided.

Received: 29 June 2024; Accepted: 29 January 2025;

Published online: 07 February 2025

## References

- Alberts IL, Mercolli L, Pyka T, Prenosil G, Shi K, Rominger A, Afshar-Oromieh A (2023) Large language models (LLM) and ChatGPT: what will the impact on nuclear medicine be? *Eur J Nucl Med Mol Imaging* 50(6):1549–1552. <https://doi.org/10.1007/s00259-023-06172-w>
- Antonenko PD, Niederhauser DS (2010) The influence of leads on cognitive load and learning in a hypertext environment. *Comput Hum Behav* 26(2):140–150. <https://doi.org/10.1016/j.chb.2009.10.014>
- Arora C, Venaik U, Singh P, Goyal S, Tyagi J, Goel S, Singhal U, Kumar D (2024). Analyzing LLM usage in an advanced computing class in India. *arXiv preprint arXiv:2404.04603*. <https://doi.org/10.48550/arXiv.2404.04603>
- Bandura A (1978) The self system in reciprocal determinism. *Am Psychologist* 33(4):344. <https://doi.org/10.1037/0003-066x.33.4.344>
- Becker BA, Denny P, Finnie-Ansley J, Luxton-Reilly A, Prather J, Santos EA (2023) Programming is hard-or at least it used to be: educational opportunities and challenges of ai code generation. *Proc 54th ACM Tech Symp Comput Sci Educ V 1*:500–506. <https://doi.org/10.1145/3545945.3569759>
- Belmar H (2022) Review on the teaching of programming and computational thinking in the world. *Front Comput Sci* 4:997222. <https://doi.org/10.3389/fcomp.2022.997222>
- Bloom BS, Engelhart MD, Furst E, Hill WH, Krathwohl DR (1956). *Handbook I: cognitive domain*. New York: David McKay
- Bryant J, Heitz C, Sanghvi S, and Wagle D (2020). How artificial intelligence will impact K-12 teachers. Retrieved May, 12, 2020
- Cao Y, Li S, Liu Y, Yan Z, Dai Y, Yu PS, Sun L (2023). A comprehensive survey of ai-generated content (aigc): a history of generative ai from gan to chatgpt. *arXiv preprint arXiv:2303.04226*. <https://doi.org/10.48550/arxiv.2303.04226>
- Chang S-C, Wongwatkit C (2023). Effects of a peer assessment-based scrum project learning system on computer programming's learning motivation, collaboration, communication, critical thinking, and cognitive load. *Education and Information Technologies*, pp 1–24. <https://doi.org/10.1007/s10639-023-12084-x>
- Chen C, Hadyong P, Brennan K, Sonnet G, Sadler P (2019) The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Comput Sci Educ* 29(1):23–48. <https://doi.org/10.1080/08993408.2018.1547564>
- Chen J, Hu L, Wu F (2023) Investigation into the transformation of knowledge-centered pedagogy with ChatGPT/generative AI. *J East China Norm Univ (Educ Sci)* 41(7):177. <https://doi.org/10.16382/j.cnki.1000-5560.2023.07.016>
- Cohen J (ED). (2013). *Statistical power analysis for the behavioral sciences*. Routledge
- Cohen SG, Bailey DE (1997) What makes teams work: group effectiveness research from the shop floor to the executive suite. *J Manag* 23(3):239–290. <https://doi.org/10.1177/014920639702300303>
- Csizmadia A, Curzon P, Dorling M, Humphreys S, Ng T, Selby C, Woollard J (2015). Computational thinking: a guide for teachers. Retrieved from Computing at Schools. website: <https://community.computingatschool.org.uk/resources/2324/single>
- Denner J, Werner L, Campe S, Ortiz E (2014) Pair programming: Under what conditions is it advantageous for middle school students? *J Res Technol Educ* 46(3):277–296. <https://doi.org/10.1080/15391523.2014.888272>
- Edmondson A (1999) Psychological safety and learning behavior in work teams. *Adm Sci Q* 44(2):350–383. <https://doi.org/10.2307/2666999>
- Fui-Hoon Nah F, Zheng R, Cai J, Siau K, Chen L (2023). Generative AI and ChatGPT: applications, challenges, and AI-human collaboration. In: vol 25. Taylor & Francis pp 277–304. <https://doi.org/10.1080/15228053.2023.2233814>
- Herm L-V (2023). Impact of explainable ai on cognitive load: Insights from an empirical study. *arXiv preprint arXiv:2304.08861*. <https://doi.org/10.48550/arxiv.2304.08861>
- Hsu Y-C, Irie NR, Ching Y-H (2019) Computational thinking educational policy initiatives (CTEPI) across the globe. *TechTrends* 63:260–270. <https://doi.org/10.1007/s11528-019-00384-4>
- Hubwieser P, Mühling A (2015). Investigating the psychometric structure of Bebras contest: towards measuring computational thinking skills. In: 2015 international conference on learning and teaching in computing and engineering, pp 62–69. <https://doi.org/10.1109/LaTiCE.2015.19>
- Hutson J, Plate D (2023). Human-AI collaboration for smart education: reframing applied learning to support metacognition. *IntechOpen*. <https://doi.org/10.5772/intechopen.1001832>
- Hwang G-J, Yang L-H, Wang S-Y (2013) A concept map-embedded educational computer game for improving students' learning performance in natural science courses. *Comput Educ* 69:121–130. <https://doi.org/10.1016/j.compedu.2013.07.008>
- Johnson DW, Johnson RT (eds). (1987). *Learning together and alone: Cooperative, competitive, and individualistic learning*. Prentice-Hall, Inc
- Kalaian SA, Kasim RM (eds). (2015). Small-group vs. competitive learning in computer science classrooms: a meta-analytic review. In *Innovative teaching strategies and new learning paradigms in computer programming*. IGI Global, pp 46–64. <https://doi.org/10.4018/978-1-4666-7304-5.ch003>
- Kasneci E, Seifler K, Küchemann S, Bannert M, Dementieva D, Fischer F, Gasser U, Groh G, Günemann S, Hüllermeier E (2023) ChatGPT for good? On opportunities and challenges of large language models for education. *Learn Individ Differ* 103:102274. <https://doi.org/10.1016/j.lindif.2023.102274>
- Liang J-C, Hwang G-J (2023) A robot-based digital storytelling approach to enhancing EFL learners' multimodal storytelling ability and narrative engagement. *Comput Educ* 201:104827. <https://doi.org/10.1016/j.compedu.2023.104827>
- Looker N (2021). A pedagogical framework for teaching computer programming: a social constructivist and cognitive load theory approach. In: *Proceedings of the 17th ACM conference on international computing education research*, <https://doi.org/10.1145/3446871.3469778>
- Louca LT, Zacharia ZC (2008) The use of computer-based programming environments as computer modelling tools in early science education: the cases of textual and graphical program languages. *Int J Sci Educ* 30(3):287–323. <https://doi.org/10.1080/09500690601188620>
- Mannila L, Dagiene V, Demo B, Grgurina N, Mirolo C, Rolandsson L, Settle A (2014). Computational thinking in K-9 education. In: *Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference*, <https://doi.org/10.1145/2713609.271361>
- Mladenović M, Boljat I, Žanko Ž (2018) Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Educ Inf Technol* 23:1483–1500. <https://doi.org/10.1007/s10639-017-9673-3>
- Nouri J, Zhang L, Mannila L, Norén E (2020) Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Educ Inq* 11(1):1–17. <https://doi.org/10.1080/20004508.2019.1627844>
- O'Hara KJ, Blank D, Marshall, J (2015). Computational notebooks for AI education. In: *The twenty-eighth international flairs conference*, pp. 263–268. [https://repository.brynmawr.edu/cgi/viewcontent.cgi?article=1030%26context=compsci\\_pubs](https://repository.brynmawr.edu/cgi/viewcontent.cgi?article=1030%26context=compsci_pubs)
- Oluk A, Korkmaz Ö (2016) Comparing students' scratch skills with their computational thinking skills in terms of different variables. *Online Submiss* 8(11):1–7. <https://doi.org/10.5815/ijmecs.2016.11.01>
- Ouaazki A, Bergram K, Holzer A (2023). Leveraging ChatGPT to enhance computational thinking learning experiences. In: 2023 IEEE international conference on teaching, assessment and learning for engineering (TALE), <https://doi.org/10.1109/TALE56641.2023.10398358>
- Rich PJ, Jones B, Belikov O, Yoshikawa E, Perkins M (2017) Computing and engineering in elementary school: the effect of year-long training on elementary teacher self-efficacy and beliefs about teaching computing and engineering. *Int J Comput Sci Educ Sch* 1(1):1–20. <https://doi.org/10.21585/ijcses.v1i1.6>
- Sharma K, Papavaslopoulou S, Giannakos M (2019) Coding games and robots to enhance computational thinking: how collaboration and engagement moderate children's attitudes? *Int J Child-Comput Interact* 21:65–76. <https://doi.org/10.1016/j.jicci.2019.04.004>
- Strom PS, Strom RD (2002) Overcoming limitations of cooperative learning among community college students. *Community Coll J Res Pract* 26(4):315–331. <https://doi.org/10.1080/106689202753546466>
- Tan SC, Chen W, Chua BL (2023) Leveraging generative artificial intelligence based on large language models for collaborative learning. *Learn Res Pract* 9(2):125–134. <https://doi.org/10.1080/23735082.2023.2258895>

- Tikva C, Tambouris E (2021) Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review. *Comput Educ* 162:104083. <https://doi.org/10.1016/j.compedu.2020.104083>
- Valmeekam K, Olmo A, Sreedharan S, Kambhampati S (2022). Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *NeurIPS 2022 Foundation Models for Decision Making Workshop*. <https://openreview.net/forum?id=wUU-7XTL5XO>
- Vasile C, Marhan A-M, Singer FM, Stoicescu D (2011) Academic self-efficacy and cognitive load in students. *Procedia-Soc Behav Sci* 12:478–482. <https://doi.org/10.1016/j.sbspro.2011.02.059>
- Wang S-L, Hwang G-J (2012) The role of collective efficacy, cognitive quality, and task cohesion in computer-supported collaborative learning (CSCL). *Comput Educ* 58(2):679–687. <https://doi.org/10.1016/j.compedu.2011.09.003>
- Wang T, Zhou N, Chen Z (2024). Enhancing computer programming education with llms: a study on effective prompt engineering for python code generation. *arXiv preprint arXiv:2407.05437*. <https://doi.org/10.48550/arXiv.2407.05437>
- Williamson B (2016) Political computational thinking: policy networks, digital governance and 'learning to code'. *Crit Policy Stud* 10(1):39–58. <https://doi.org/10.1080/19460171.2015.1052003>
- Wing JM (2006) Computational thinking. *Commun ACM* 49(3):33–35. <https://doi.org/10.1145/1118178.1118215>
- Wu J, Gan W, Chen Z, Wan S, Lin H (2023). Ai-generated content (aigc): a survey. *arXiv preprint arXiv:2304.06632*. <https://doi.org/10.48550/arxiv.2304.06632>
- Xinogalos S, Satratzemi M, Chatzigeorgiou, A, Tsompanoudi D (2017). Student perceptions on the benefits and shortcomings of distributed pair programming assignments. In: 2017 IEEE global engineering education conference (EDUCON), pp 1513–1251. <https://doi.org/10.1109/EDUCON.2017.7943050>
- Yildiz Durak H (2018) Digital story design activities used for teaching programming effect on learning of programming concepts, programming self-efficacy, and participation and analysis of student experiences. *J Comput Assist Learn* 34(6):740–752. <https://doi.org/10.1111/jcal.12281>
- Yilmaz R, Yilmaz FGK (2023a) Augmented intelligence in programming learning: examining student views on the use of ChatGPT for programming learning. *Comput Hum Behav: Artif Hum* 1(2):100005. <https://doi.org/10.1016/j.chbah.2023.100005>
- Yilmaz R, Yilmaz FGK (2023b) The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Comput Educ: Artif Intell* 4:100147. <https://doi.org/10.1016/j.caeai.2023.100147>
- Yuliyanto A, Turmudi T, Agustin M, Putri HE, Muqodas I (2019) The interaction between concrete-pictorial-abstract (CPA) approach and elementary students' self-efficacy in learning mathematics. *Al Ibtida: J Pendidik Guru MI* 6(2):244–255. <https://doi.org/10.24235/al.ibtida.snj.v6i2.5226>
- Zhan Z, He G, Li T, He L, Xiang S (2022) Effect of groups size on students' learning achievement, motivation, cognitive load, collaborative problem-solving quality, and in-class interaction in an introductory ai course. *J Comput Assist Learn* 38(6):1807–1818. <https://doi.org/10.1111/jcal.12722>
- Zhang L, KaLyuga S, Lee C, Lei C (2016) Effectiveness of collaborative learning of computer programming under different learning group formations according to students' prior knowledge: A cognitive load perspective. *J Interact Learn Res* 27(2):171–192
- Zhang P, Tur G (2023). A systematic review of ChatGPT use in K-12 education. *Eur J Educ*. <https://doi.org/10.1111/ejed.12599>
- Zheng L, Zhen Y, Niu J, Zhong L (2022) An exploratory study on fade-in versus fade-out scaffolding for novice programmers in online collaborative programming settings. *J Comput High Educ* 34(2):489–516. <https://doi.org/10.1007/s12528-021-09307-w>
- Zheng L, Zheng L (2021). Improving programming skills through an innovative collaborative programming model: a case study. In: *Data-driven design for computer-supported collaborative learning: design matters*, pp 75–85. [https://doi.org/10.1007/978-981-16-1718-8\\_6](https://doi.org/10.1007/978-981-16-1718-8_6)

## Author contributions

Yi-Miao Yan is the first author, Chuang-Qi Chen is the second author, and Yang-Bang Hu and Xin-Dong Ye are the co corresponding authors of this article. Yi-Miao Yan, Chuang-Qi Chen, Yang-Bang Hu, and Xin-Dong Ye had designed and conducted the research. Yi-Miao Yan and Chuang-Qi Chen had carried out the data collection, inspection, and analysis. Yi-Miao Yan, Yang-Bang Hu, and Xin-Dong Ye had created the data processing charts. Yi-Miao Yan and Chuang-Qi Chen had written the paper, while Yang-Bang Hu had proofread the draft and provided guidance on important decisions. Xin-Dong Ye had provided full text review guidance for the paper, addressing difficult and complex problems. All authors had contributed to each revision of the article.

## Funding

The author declare financial support was received for the research, authorship, and/or publication of this article. The work of Yi-Miao Yan was Supported by the Graduate Scientific Research Foundation of Wenzhou University (3162024003019).

## Competing interests

The authors declare no competing interests.

## Ethical approval

This work involved human subjects or animals in its research. This study was performed in line with the principles of the Declaration of Helsinki. Approval was granted by the Ethics Committee of Wenzhou University (Date: February 23,2024/No: WZU-2024-107).

## Informed consent

This study has obtained informed consent from all participants and/or their legal guardians to ensure their understanding of the purpose, content, methods, potential risks, and benefits of this study, and to allow the research team to use and publish the data, relevant images, and audio generated during the research process.

## Additional information

**Supplementary information** The online version contains supplementary material available at <https://doi.org/10.1057/s41599-025-04471-1>.

**Correspondence** and requests for materials should be addressed to Yang-Bang Hu or Xin-Dong Ye.

**Reprints and permission information** is available at <http://www.nature.com/reprints>

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025

## Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).

Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.

These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.

We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.

While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval, sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.

These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.

To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.

Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.

If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

[onlineservice@springernature.com](mailto:onlineservice@springernature.com)