Journal of Theoretical and Applied Information Technology
31st May 2025. Vol.103. No.10
© Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REVIEW ON PROGRAMMING LANGUAGE LEARNING MODELS AND INSTRUCTIONAL APPROACHES: CURRENT TRENDS AND FUTURE DIRECTIONS

SANAL KUMAR T S[1] AND R THANDEESWARAN[1*]

[1]School of Computer Science Engineering and Information Systems, Vellore Institute of Technology (VIT), Vellore, 632014, Tamil Nadu, India.

[*]Corresponding author. E-mail: rthandeeswaran@vit.ac.in;

Contributing author: thesanals@gmail.com

ABSTRACT

Programming has become a fundamental subject in the academic curriculum, but the learning process presents unique challenges. It requires not only systematic study and dedication but also the application of logical thinking and problem-solving skills in practical contexts. This complexity makes programming particularly difficult for beginners, who often face challenges in understanding foundational concepts like sequencing, decision-making, and looping. As a result, various pedagogical methods have evolved to address the difficulty in programming learning. To better understand the recent developments in programming educational approaches, we aim to provide a detailed review of learning models and instructional approaches in programming learning. Following this, we explore the cognitive factors influencing the learner and the essential aspects of the learner's learning style and preferences in programming education. Finally, we conclude the review by discussing how these techniques can be combined to formulate future pedagogical approaches in programming instruction. Consequently, this review proposes integrating the learning style model with adaptive e-learning environments (ALE) in a blended learning approach as a better solution to address the hurdles of programming learning difficulty. Given this, the review paper provides a comprehensive overview of the programming learning environments, strategies, instructional approaches, cognitive factors, and learning styles leveraged in programming education, which future researchers can utilize.

Keywords: *Programming Education; Difficulty In Programming; Instructional Methods; Learning Style Models; Adaptive Learning Environments.*

1. INTRODUCTION

Programming has become an essential part of primary and higher education, extending beyond traditional computer science fields to basic sciences and management. Despite its importance, learning to program poses unique challenges, as it demands not only theoretical understanding but also practical application of concepts through logical thinking and problem-solving. Core programming constructs like sequencing, decision-making, and looping are particularly challenging for beginners, who often struggle to grasp these foundational elements. This difficulty has led to high dropout and failure rates in introductory programming courses, which has, in turn, sparked significant research into better instructional methods for programming education [1].

In response to these challenges, researchers have proposed various teaching and learning methods to make programming more accessible, incorporating diverse learning models and instructional strategies. Although some methods have shown promising results, there is still a lack of consensus on the most effective approaches [2]. This review investigates current trends in programming education, exploring the impact of different learning models, instructional approaches, and cognitive factors that shape how students learn programming. By analyzing these elements, the review aims to clarify which approaches have the greatest potential to improve learning outcomes and student retention in programming courses.

This review differs from previous works in both motivation and approach. While prior studies have focused on evaluating isolated techniques or proposing specific frameworks, this work aims to provide a comprehensive evaluation of existing learning models and instructional strategies in programming education. Unlike earlier reviews that primarily catalog methodologies, this study emphasizes the individual preferences and cognitive barriers faced by learners and highlights how these factors influence learning outcomes. Furthermore, it explores the integration of adaptive learning technologies with traditional methods to better address the diverse needs of learners.

The primary objective of this study is to examine existing models and methods used to teach programming and assess their effectiveness within current educational trends. Additionally, this review aims to identify the key challenges in programming instruction, such as cognitive and motivational barriers, and propose directions for future research. This includes exploring integrative instructional strategies that may address the limitations of existing models and meet the diverse needs of learners in a rapidly evolving digital environment. In doing so, the review provides insights into how programming education can be improved to support students more effectively, potentially reducing dropout rates and fostering a deeper understanding of programming principles.

1.1. Programming Learning Difficulty

Difficulty in learning programming is a universal scenario. Researchers reported several causes for difficulty in programming learning. The most critical issue is the lack of problem-solving abilities, which most students lack. Learning and writing computer programs can be particularly difficult for students who have not been properly introduced to the fundamental concepts and skills required. Without a strong foundation, students may struggle with understanding the syntax and logic of programming languages, debugging errors, and applying theoretical knowledge to practical problems. The lack of skilled instructors to teach programming is an issue, as well as a lack of scientific methods or techniques to learn the art of programming [3]. Instructors who teach programming language subjects have less pedagogical support to prepare and practice computer programming, especially when the teacher needs to teach it as an introductory course. Researchers propose different methods to address these problems, which may vary depending on the learner's learning capability.

Learning programming can be challenging for several reasons. First, programming requires a strong foundation in logic and problem-solving skills, which can take time for beginners to develop. The abstract nature of programming concepts, such as algorithms, data structures, and syntax, often leads to confusion and frustration. Additionally, debugging and troubleshooting code demand patience and a systematic approach, skills that take time to cultivate. A learning curve focuses on programming learning, which estimates a learner's performance over time. The steep learning curve is further compounded by the need to understand the intricacies of different programming languages and tools. Finally, the lack of immediate feedback and personalized guidance in traditional learning environments can hinder progress, making it hard for students to identify and correct their mistakes promptly. These factors combined make programming a demanding subject for mastery [3].

1.2. Objective of The Study

a. To analyze and understand diverse learning models, instructional strategies, and cognitive elements that influence programming education, focusing on approaches that support novice learners and enhance programming comprehension and retention.

b. To identify and assess key gaps in the literature related to existing instructional methods, learner engagement strategies, and adaptive learning models in programming, particularly those that fail to account for individual learning preferences and the unique challenges posed by programming for beginners.

c. To propose future directions and improvements by integrating insights from recent technological advancements, such as adaptive and blended learning approaches, that address the limitations found in current programming instruction and better support diverse learner needs and cognitive differences.

1.3. Outline

This paper is organized as follows: Section I covers the significance and challenges of introductory programming courses. Section II provides an in-depth review of the literature on learning environments, strategies, instructional approaches, and cognitive factors relevant to programming education, while also highlighting traditional learning style models. Section III identifies gaps within the existing research, and Section IV outlines the research methodology used in this study. Section V presents current trends and suggests future directions. Section VI discusses several critical challenges remain unaddressed in programming education, followed by limitation in the study, and finally, Section VI offers the conclusion. The schematic arrangement is depicted in Fig.1.

2. LITERATURE REVIEW

In this study, a selective review of the existing literature has been conducted in alignment with the research framework's objectives. This

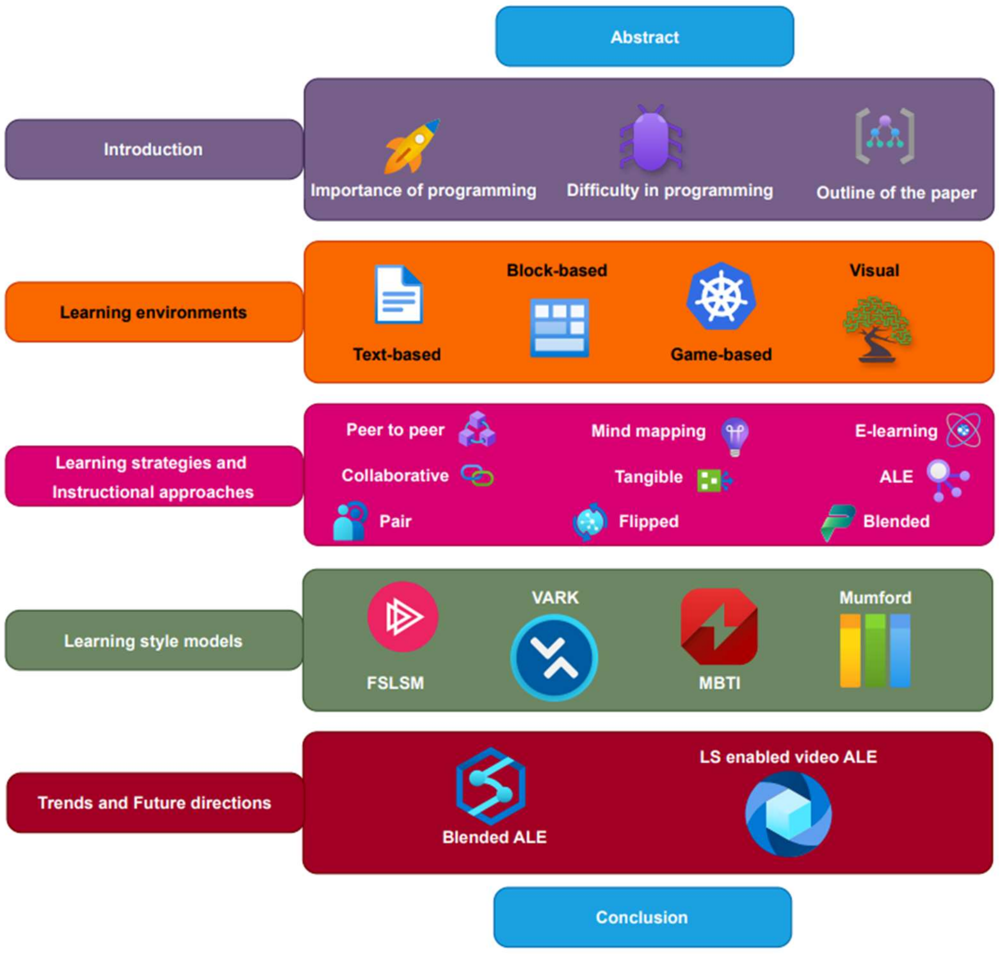

Figure 1: The schematic arrangement of this paper

approach aids in identifying unresolved gaps within the selected research area and offers suggestions for future research directions.

2.1. Programming Learning Environments

Researchers and developers build various learning tools and interfaces to address the difficulties in learning programming. The aim is to ease the learning process and increase student success rate. Text-based, Block-based, game-based, and visual programming are programming environments designed to teach and facilitate coding skills, particularly for beginners or young learners [4]. Both text-based and block-based programming environments are educational tools that cater to different learning stages and preferences. Block-based programming is often used in introductory programming courses for children and beginners, as it visually represents the flow of logic without requiring detailed knowledge of syntax. Text-based programming, on the other hand, is used for more advanced programming education and professional development, requiring learners to understand and use the precise syntax of a programming language [2].

Game-based programming involves learning to code through interactive and engaging game scenarios. These environments are designed to teach programming concepts by integrating them into game design and gameplay [5]. This method is highly motivational for learners, as it combines the fun of gaming with the educational aspects of coding. It helps learners understand programming logic and problem-solving skills within a gamified context. On the other hand, visual programming environments allow users to create programs by manipulating elements graphically rather than writing code textually. These environments are designed to simplify the coding process by enabling users to drag and drop visual elements that represent code structures and logic. Moreover, visual programming benefits beginners, young learners, and those who benefit from a more intuitive and visual approach to understanding programming concepts with ease [6]. In the following sub-sections, we summarized different studies related to Text-based, Block-based, game-based, and visual programming environments.

2.1.1. Text-based programming

In the early days, only text-based programming languages were used to teach programming. Programming learning with High-level programming languages started with text-based procedure-oriented languages such as Fortran, Cobol, and Algol [2]. Text-based programming environments can be either text editors or IDEs (Integrated development environments). Because of its unfriendly nature, it was difficult for the average student to learn and write programs legitimately. The lack of pedagogical methods made the process very cumbersome. Novice learners are not expected to learn programming with text-based programming environments since their first experience confirms it. Simpler programming languages and environments are advisable because they are more approachable. Nowadays, programming starts in the early stages of education. The complicated and sophisticated programming constructs confuse the novice programmer and negatively affect the learning process.[2].

2.1.2. Block-based programming

Traditional programming instruction often relies on text-based programming environments. However, the syntactic and semantic challenges associated with text-based coding have driven the development of block-based and game-based programming approaches. For novice learners, block-based programming languages are more convenient and easy to learn, moreover, they help them to reduce the programming learning barrier [7]. The first modern block-based programming language was AgentSheets [4], designed for kids to learn programming hassle-free. It is a game-based learning platform that uses a block-based programming method. AgentSheet was designed by Dr. Alexander Repenning as a research product to enhance computational thinking among children [3]. StarLogo is a visual and 3D environment that uses block coding to help novice learners model simple games and learn programming quickly. Scratch is another block-based visual programming environment that allows students to develop interactive media-rich projects. Scratch is getting much attention among online coding platforms for kids and young students. It is designed as a single-window, multi-pane interface that makes it much more user-friendly to novice programmers and ensures that all major components are easily accessible. All commands are readily accessible from the command palette, which includes different categories like Motion, Looks, Sound, control, etc. [3]. Scratch is also used in Dynamic Programming (DP) to solve large problems that computer science students worldwide find hard to understand. The problem with pure block-based programming learners is that writing text-based programs requires additional effort and time, which are significantly different from the block-based environment. Block-C is another block-based language based on C programming language that empowers novice students to learn programming by focusing only on logic rather than complicated syntax. The advantage of Block C is that the transition facility from textual C to Block C and Block C to textual C is straightforward. Most educators suggest Block C because all other introductory Block-based languages like Scratch and Star Logo are treated as toy-like structures without real connection with C, Java, or Python [7]. Due to its ease of use and reduced complexity, instructors often choose block-based programming environments as a starting point for novice learners.

2.1.3. Game-based programming

Game-based programming involves learning to code by immersing students in interactive and engaging game scenarios. This method leverages the mechanics and elements of games to teach programming concepts in a way that is both enjoyable and educational. These studies [5, 8–14] collectively explore the integration and impact of game-based learning (GBL) in programming education across various levels. In this paper [8], the' PROBSOL' application was introduced to enhance problem-solving skills in novice programmers, and survey results show positive feedback from both genders, emphasizing the applications' user-friendliness and efficiency. Another study [9] focuses on GBL's potential to improve student engagement and programming skills, with findings supporting the effectiveness of educational games in fostering enjoyment and learning. In this work [10], they aim to enhance a Python programming course by combining innovative game modules with traditional teaching methods, emphasizing motivation and competency improvement. Another study [11] explores the integration of game-based learning and game design as a novel teaching method for programming in primary and secondary schools. The study aims to fill the gap in information regarding game design's specific role in teaching programming, focusing on novice learners. In this study [5], they developed ZTECH, a proprietary game-based learning tool designed to enhance students' understanding of object-oriented programming. This role-playing game offers an interactive environment with quests and challenges, fostering self-motivation and an engaging learning experience for object-oriented programming concepts.' Programmer Adventure Land,' a game-based learning courseware [12], employs a problem-based strategy to enhance college students' understanding of computer programming. The results indicate that the problem-based learning approach improves satisfaction, enjoyment, motivation, and user interface in learning computer programming. Additionally, initiatives like' CODING4GIRLS' [13] use game development for programming skill cultivation, showing positive acceptance among students in lower secondary education. Lastly, a study [14] examines the intentions of younger adolescents toward game-based programming learning, identifying key factors and moderating effects of gender and grade level on students' attitudes. Overall, these studies underscore the valuable role of GBL in diverse programming education contexts. However, game-based programming can be challenging because it requires good game design. At the same time, it can distract from core programming concepts and may not work for all learning styles.

2.1.4. Visual programming

Visual programming environments (VPEs) allow users to create programs by manipulating graphical elements instead of writing text-based code. These environments provide a user-friendly interface where visual icons, blocks, or diagrams represent programming concepts. NetsBlox is a visual programming environment designed for learning distributed programming principles, offering accessible abstractions like message passing and Remote Procedure Calls. The study [15] examined the effectiveness of a Visual Programming Language (VPL) intervention, finding that it significantly improved students' grasp of fundamental programming concepts, particularly benefiting those without a computer science background. This paper [16] conducts a meta-analysis of 29 studies, revealing the positive impact of block-based visual programming tools on students' academic achievement, with influencing factors identified. This article [17] investigates challenges in programming education, focusing on object-oriented programming, and suggests the need for a comprehensive learning environment. In this work [18], they emphasize the importance of addressing algorithmic thinking and problem-solving in programming education, recommending Scratch as a suitable visual programming language for beginners. This research [6] focuses on teaching programming to primary school children through online interactive environments, demonstrating positive attitudes and improved problem-solving skills. The study [19] explores the impact of teachers' pedagogical perspectives on integrating visual programming in early formal education, emphasizing the potential promotion of constructivist pedagogy. Lastly, the study [20] investigates the correlation between students' satisfaction levels with visual learning environments (Greenfoot and Alice) and their academic performance in programming courses, revealing a significant positive correlation and highlighting the impactful role of these environments in improving students' outcomes compared to traditional methods. While VPEs simplify complex concepts by using graphics and visual blocks instead of traditional text-based syntax, they may not cater to all learning styles effectively. VPEs are particularly beneficial for visual learners who process information best when it is presented in a graphical format, as well as for kinesthetic learners who benefit from hands-on, interactive tasks. Therefore, while VPEs can be valuable for introducing programming concepts, they might need to be supplemented with other instructional methods to accommodate the diverse needs of all learners.

2.2. Programming Learning Strategies and Instructional Approaches

Researchers have developed a variety of innovative instructional approaches to teach programming effectively. This section examines several instructional methods for programming education (Fig. 2), organized into distinct categories as outlined below.

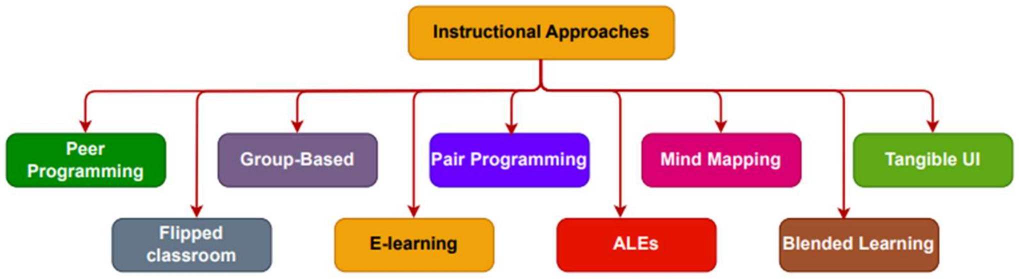

Figure 2: Learning strategies and Instructional approaches

2.2.1. Peer programming (Peer assisted learning) strategy

Peer-assisted learning [21] is a concept to help students each other in the learning process. The learning group consists of two students who are "coach" and "learner" with instructor support at any time. In this study [22], mobile, agile peer assisted learning was used to overcome the difficulties of programming learning. They designed a platform to support programming learning using C++. Here [23], students learned an introductory course on Arduino, which uses a peer-based learning technique. A peer learning agent [21] acts as an aid to learning programming and analyzes the learning levels of the student. They used the Bayesian network and programming pedagogical methods to simulate this system. [24] used the C programming language PAL (Peer-assisted learning environment) to learn programming and tested it with experimental and control groups. All the above work systematically claims that peer-assisted learning improves the interest and learning capability of the students. The main drawback of this technique is the inability of the peer learning agent to understand learner specific preferences and learning characteristics of the learner.

2.2.2. Group-based or Collaborative learning

Collaborative learning, or group-based learning, is a student-centric approach that empowers students to participate actively in the course material and with each other [25]. It involves students working in teams or groups, promoting interaction, discussion, and joint problem-solving. Through this active engagement, students enhance their understanding of content and develop crucial interpersonal and teamwork skills, taking ownership of their learning journey [26].

This paper [27] discusses redesigning an introductory programming course using the community of inquiry learning framework and design patterns from online communities and team-based learning. An experimental evaluation involving 562 students indicates that the experimental groups outperformed the control group, with the CoL (community of inquiry learning framework) and TBL (team-based learning) methodology leading to higher levels of understanding due to increased participation rates. A real-time collaboration version of MIT App Inventor (MAI) was proposed [28] to facilitate cross-region and multi-user collaborative software development. An empirical study compared self-efficacy and collaborative behavior of learners using MAI with and without real-time collaboration, finding that engagement in joint behavior increased with real-time collaboration, particularly among CS-major groups. In this study [29], they aim to improve computer programming skills through collaborative learning with a problem-based practice strategy. Results from testing with two classes show that the proposed strategy enhances students' programming skills in a collaborative learning environment. In this paper [30], they introduce a novel computer-supported collaborative learning group designed for problem-based collaborative learning in computer programming education. The system enables learners and tutors from different locations to collaboratively address programming challenges using synchronous and asynchronous tools within a shared workspace. The study outlines the groupware's functionalities and conducts an experimental evaluation, applying the unified technology acceptance theory, to assess learners' Behavioral Intention (BI) within the context of Algerian higher education. The primary limitation of this collaborative learning is its inflexibility to recognize and adapt to personalize learner demands and requirements.

2.2.3. Pair programming

Pair programming is a teamwork approach where two individuals share a computer while working together to develop software. While it has been employed in industry, its popularity has grown significantly in educational contexts [31]. In introductory programming courses, pair programming is commonly utilized because there is substantial evidence that it enhances students' learning of programming concepts. Studies [31, 32] indicate that pair programming offers advantages such as improved success rates in introductory courses, higher retention within majors, enhanced software quality, increased student confidence in solutions, and better learning outcomes. Additionally, evidence suggests that women, in particular, benefit from pair programming. Furthermore, the transition from paired to solo programming appears to be straightforward for students, although scheduling and partner compatibility remain significant challenges.

2.2.4. Mind mapping teaching

A mind map is a diagram that organizes information into a hierarchy, revealing relationships among various elements. It can be used to assist novice learners in teaching programming. Mind maps are helpful for creativity, understanding programming challenges, and designing solutions. The integration of mind mapping into programming education has garnered substantial attention, yielding valuable insights across diverse contexts [33]. An experimental study explored the effectiveness of using Mind Maps as a brainstorming and conceptualization tool for programming. The results were compelling: employing mind maps within text- and block-based programming environments significantly improved student's learning outcomes. Moreover, researchers explicitly used Scratch to investigate the impact on computational thinking (CT) skills when integrating mind mapping into programming language instruction [34].

Comparing two mind mapping approaches-construct-on-scaffold (COS-MM) and construct-by-self (CBSMM)-the study [35] revealed that COS-MM was more effective in enhancing student's CT skills. Furthermore, in the context of primary school instruction, mind mapping facilitated programming comprehension and promoted creativity among students engaged in the Scratch course. Beyond primary education, a study in Malaysia explored the effects of mind mapping combined with cooperative learning (MMCL) versus traditional cooperative learning (CL). The striking results: MMCL significantly improved programming performance and enhanced students' metacognitive knowledge [36]. Lastly, an experimental study [37] focused on modern programming languages, concluding that incorporating mind-mapping tools enhances undergraduate students' critical thinking and problem-solving skills. In summary, mind mapping is a versatile pedagogical approach fostering creativity, cognition, and collaboration in the dynamic landscape of programming education. A limitation of mind mapping is that it doesn't account for basic learning habits or individual preferences, making it difficult for learners to select learning materials that align with their understanding styles.

2.2.5. Tangible user interface

Tangible User Interfaces (TUIs) represent an innovative interface paradigm that effectively addresses the limitations of traditional Graphical User Interfaces (GUIs). Unlike GUIs, which rely on mouse and keyboard input, TUIs allow users to interact with systems by manipulating real-life physical objects. Research indicates that the tangibility and hands-on manipulation of physical elements enhance performance, learning, decision-making, and user retention, leading to a more substantial learning gain. Furthermore, using tangible interfaces injects playfulness into problem-solving, making the learning experience more engaging [38]. This section explores the potential of tangible user interfaces (TUI) to enhance the learning of abstract programming concepts, focusing on a study [38] that investigated the usability of a TUI system designed for teaching basic Java programming. The researchers developed a prototype using Processing and ReacTI Vision and assessed its usability with the System Usability Scale. While the system showed acceptable usability, identified limitations highlight areas for further improvement in TUI-based programming education. Addressing the growing demand for programming skills starting at a young age, this study [39] examines the impact of tangible programming interfaces compared to traditional visual methods, particularly for children around six. The study showcases the" Follow Your Objective" (FYO) platform, a cost-effective tangible programming solution featuring an intuitive programming board, puzzle-based tangible blocks, and a mobile robot. Preliminary experiments with FYO demonstrated enhanced programming skills among children, indicating the effectiveness of tangible puzzle-based platforms for early programming education. This work [40] discusses strategies to facilitate the learning of recursion, a complex programming technique, especially in functional languages. A proposed solution involves an interactive interface based on a tangible block world with augmented reality and software feedback, using stack blocks as analogies for list data structures. The goal is to enable students to intuitively grasp recursive concepts and transition to writing recursive programs in sequential Erlang, promoting effective recursion learning within programming education. TUIs are mainly designed to support visual and kinesthetic learners, providing interactive, hands-on learning experiences. However, they often overlook the preferences and learning styles of traditional, more text-based learners.

2.2.6. Flipped classroom-based collaborative learning

A Flipped Classroom (FC) reverses the traditional learning model by having students review instructional materials at home and engage in interactive, hands-on activities in class. The efficacy of the FC teaching method in computer programming education has been the subject of several research studies. Here is an overview of these studies and their key findings: Research studies have extensively examined the efficacy of the FC teaching method in computer programming education. One study combined FC with Problem-based Learning (PBL), finding significant improvements in students' understanding of programming concepts, particularly benefiting weaker students and making learning enjoyable for mature students [41]. Another mixed-method study revealed that students in a flipped classroom exhibited higher academic achievements and more positive attitudes toward programming. However, challenges such as technological requirements and low attendance were noted [42]. Additional research demonstrated that an adapted FC approach positively impacted programming success and self-efficacy, although it did not significantly affect attitudes toward programming [43]. A study focusing on self-regulation found varying levels of self-regulated learning skills among students, emphasizing the potential of problem-based activities in flipped learning [44]. Another investigation showed a positive change in students' acceptance of programming after implementing the FC model, with students agreeing that it improved their learning experience [45]. Furthermore, a comparative study revealed that students in a flipped classroom achieved higher test scores and had positive perceptions of in-class activities [46]. Lastly, research exploring students' perspectives on enriching programming and algorithm teaching with the FC approach found that most students had positive views, highlighting its potential effectiveness in programming courses [47]. These studies [41-47], collectively suggest that the flipped classroom approach holds promise for improving programming education outcomes, enhancing student engagement, and fostering positive attitudes toward programming. However, problems such as technology integration and attendance must be solved to fully realize this teaching technique's benefits.

2.2.7. Programming using e-learning

Programming education is crucial for students in various disciplines, necessitating effective learning strategies and motivation. Several studies have explored innovative approaches and e-learning systems to enhance motivation and learning outcomes in programming courses. Here's an overview of these studies: The study [48] investigates motivating factors influencing undergraduate students' learning in computer programming courses. The study highlights the Programming Assignment aSsessment System (PASS), an e-learning infrastructure to support programming education. Key motivating factors include individual attitude, clear direction, and reward/recognition. The study suggests that well-facilitated e-learning environments can enhance motivation and self-efficacy. This study [49] addresses challenges in distance and e-learning for programming subjects by proposing virtual pair programming (VPP). The research focuses on asynchronous VPP and assesses its effectiveness in teaching object-oriented programming at Open University Malaysia (OUM). Positive feedback from learners suggests the potential of asynchronous VPP, with suggestions for further enhancements. This study [50] highlights the need for improving conceptual learning in basic computer programming using personalized e-learning environments. The study incorporates personalized information like learning problems, styles, and performance levels to tailor the learning experience. Results indicate that students using the personalized e-learning environment demonstrated improved understanding and positive attitudes toward programming.

The work [51] develops a PROBSOL application to enhance novice programmers' problem-solving skills in introductory programming courses. The study compares web-based and mobile app versions of PROBSOL and assesses their impact on student engagement and learning outcomes. Results show improved cognitive gains, logic capabilities, and reduced attrition rates using PROBSOL. The study [52] proposes an e-learning model for programming instruction to secondary school students to enhance motivation and understanding. The model emphasizes collaborative learning to address the challenges of learning ICT subjects. The study aims to meet the demand for informatics competencies in secondary school curricula. This study [53] introduces interactive multimedia e-learning to help undergraduate students grasp the fundamental concepts of logic and algorithms in programming. The study addresses the limitations of traditional e-learning systems and focuses on enhancing independent learning through multimedia materials and interactive exercises. This study [54] explores the impact of interactive instructions in e-learning on the effectiveness of a programming course, particularly during the COVID-19 pandemic. The study assesses usability and efficacy using a questionnaire-based evaluation method to improve e-learning systems for programming education. These studies [51-54] underscore the importance of motivation and innovative e-learning approaches in programming education. Personalized e-learning environments, virtual pair programming, and problem-solving applications demonstrate promising outcomes for improving student engagement, understanding, and learning outcomes in programming courses. Further research and enhancements are required in e-learning systems focusing on integrating learning styles and preferences in programming learning, which can significantly contribute to effective programming education.

2.2.8. Adaptive e-learning environments (ALEs)

ALEs customize educational content and activities to match each learner's preferences, pace, and learning style for a personalized experience. This section presents various adaptive learning systems tailored for computer programming instruction. This study [55] is based on the Revised Bloom Taxonomy (RBT) learning theory, incorporates fuzzy weights to convert students' knowledge levels, and employs rule-based decision-making to offer personalized learning activities in a C# programming language course. This work [56] introduces an adaptive algorithm for Java programming learners, guiding them through teaching materials and practical tasks, with an analysis demonstrating its potential. This study [57], FuzKSD, utilizes fuzzy cognitive maps and overlays to model and update a student's knowledge level dynamically, offering individualized adaptive advice in web-based education. ELaCv2 [58] integrates a 4-parameter student model for personalized adaptation in e-training for programming and C. This work [59], an Intelligent Tutoring System (ITS), combines fuzzy logic and machine learning to adapt learning material and sequences based on student knowledge. It shows improved personalized learning experiences and reduced dropout rates through evaluation. A major drawback in existing studies [55-59] is the lack of a novel learning style model tailored to meet the unique demands of programming and problem-based learning.

2.2.9. Blended learning

Blended learning has emerged as a transformative approach in programming education, especially amidst the challenges posed by the COVID-19 pandemic. This educational model combines traditional face-to-face instruction with online learning components to optimize learning outcomes and enhance students' engagement and flexibility. The research papers discussed herein explore various aspects of blended learning in programming education, from analyzing student behaviors and performance to redesigning courses and evaluating the effectiveness of blended learning models. Each study contributes unique insights and methodologies that shed light on the implementation and impact of blended learning strategies in diverse educational contexts.

Research studies have explored various aspects of blended learning in programming education, revealing insightful findings on its efficacy and challenges. [60] analyzed students' test scores and online learning behaviors in a blended programming environment, identifying distinct student types and suggesting a negative correlation between online time and test scores. This study concludes that long-term online learning stabilizes test scores and provides personalized guidance for educators. In [61], they introduced a personalized course model using Bayesian networks in Python programming education, demonstrating its effectiveness and emphasizing high satisfaction and success rates during COVID-19. In [62], they adapted Hadjerrouit's model to identify the construction phase of blended learning as significantly impacting student outcomes. The work [63] addressed the challenges of the COVID-19 pandemic by discussing evidence-based pedagogical approaches in blended learning, highlighting the benefits of face-to-face interactions and online flexibility. [64] explored professional development for teachers in programming education, finding that a blended PD program supports collaborative learning and transformation in teaching practices. [65] described redesigning an introductory computer science course using blended learning, reporting increased student engagement, lower dropout rates, and improved exam results. It also highlighted the integration of face-to-face and online learning in a specific course, showing that the blended model enhances students' control over their learning and improves academic achievements. [66] examined students' opinions on a blended learning environment, noting positive feedback on learning facilitation, interaction opportunities, and supportive course materials while addressing challenges such as insufficient teaching time and technical issues. Blended learning, while beneficial, falls short in delivering personalized learning objects that align with individual learning preferences. Current research lacks examples of blended approaches that incorporate video-based learning objects designed with learner style models in mind. This gap highlights the need for a more tailored approach that combines blended learning with adaptive content delivery based on learning styles.

In conclusion, various programming learning strategies and instructional approaches have been outlined above. A key limitation across these methods is the lack of capability to identify and adapt to individual learner preferences and unique learning characteristics. These approaches assume a one-size-fits-all method, overlooking that learners have distinct ways of absorbing information, varying levels of prior knowledge, and unique cognitive and perceptual strengths. Without accounting for these differences, these technique fails to provide tailored support that could enhance learner engagement, comprehension, and retention. Consequently, it may not effectively address diverse learning needs, which can hinder students' overall understanding and performance, especially in complex subjects like programming where adaptability to individual learning styles could be highly beneficial (table 1). Addressing these shortcomings offers a significant opportunity to improve the effectiveness and personalization of programming education.

2.3. Cognitive Factors in Learning Programming

In programming education, understanding the cognitive factors influencing learning is crucial for designing effective instructional methods and supporting student success [67, 68]. Cognitive processes play a fundamental role in how individuals acquire and apply programming knowledge, ranging from problem-solving strategies to developing algorithmic thinking skills. Mastering programming skills requires memorizing syntax and engaging in complex cognitive processes [1, 69]. This section delves into the mental aspects of learning to program, exploring how factors like problem-solving, mental models, and prior knowledge influence a student's ability to grasp and apply programming concepts. By understanding these cognitive factors, educators can develop more effective teaching strategies and learning environments that foster successful programmers.

Extensive research [1, 15, 67-73] has investigated the impact of cognitive factors on academic achievement, with particular emphasis on first-year students. This focus is driven by findings indicating that many students leave university within their first academic year or before their second due to poor academic performance. Such attrition is a significant concern for academic institutions. Studies have identified various cognitive factors that affect academic success, including

Table 1: Research Gaps In Different Programming Learning Instructional Approaches.

Authors	Programming Learning Strategies	Research Gaps
[21-24]	Peer Programming strategy	Inability to adapt to learner-specific preferences and individual learning characteristics.
[25-30]	Group-Based or Collaborative Learning	Inflexibility to recognize and adapt to personalize learner demands and requirements.
[31,32]	Pair Programming	The transition from paired to solo programming is relatively straightforward for students, significant drawbacks include challenges related to scheduling and ensuring partner compatibility.
[33-37]	Mind Mapping Teaching	Inability to account for basic learning habits and individual preferences, making it challenging for learners to choose materials that suit their understanding styles.

[38-40]	Tangible user interface	Primarily designed to support visual and kinesthetic learners by offering interactive, hands-on experiences, they often neglect the preferences and learning styles of traditional text-based learners.
[41-47]	Flipped classroom-based collaborative learning	Technology integration and attendance need to be addressed to fully harness the benefits of this teaching technique.
[48-54]	Programming using E-learning	Limited integration of learning styles and preferences, hindering their effectiveness in programming education.
[55-59]	Adaptive e-learning environments (ALEs)	The absence of a novel learning style model specifically designed to address the unique requirements of programming and problem-based learning.
[60-66]	Blended Learning	Lacks blended approaches using video-based learning objects designed with learner style models, highlighting the need for adaptive content delivery tailored to learning styles.

mental models, self-efficacy beliefs, motivation, and personality traits. Self-efficacy and motivation have consistently emerged as strong predictors of academic performance. Research, such as that referenced in [70], underscores that students with high self-efficacy are likelier to achieve academic success than those with lower self-efficacy. Additionally, [15] emphasizes the influential role of self-efficacy, positing that individuals' beliefs about their capabilities shape their anticipatory behaviors and mental rehearsals.

Investigations into these factors, such as that detailed in [71], have shown that mathematics self-efficacy is a crucial predictor of problem-solving ability. However, [71] did not explore the combined effects of these factors. In line with this, [72] suggests that self-efficacy tends to increase with prior experience. A separate study by [73] examined the influence of self-efficacy, mental models, and prior programming experience, revealing a significant increase in self-efficacy among students with previous programming exposure during their first-year programming course. As a result, this illustrates the complex interplay between cognitive factors and academic performance in higher education settings.

2.4. Learning Style Models

Learning style is a crucial element of personalized learning, as it identifies learners' characteristics and their preferred ways of learning in predefined dimensions or classes. Programming can be particularly challenging for beginners due to various factors. Today, online e-learning platforms offer programming courses, but they often deliver content in a uniform manner, assuming all learners progress at the same pace. However, each learner has their learning speed, which conventional e-learning platforms typically overlook. According to [74], learning style is defined as the "attitudes and behaviors which determine an individual's preferred way of learning." Researchers have proposed various models to address learning styles. Among the 71 identified learning styles, the Felder-Silverman Learning Style Model(FSLSM), the VARK (Visual, Auditory, Read/Write, and Kinesthetic) model, the Myers-Briggs Type Indicator (MBTI) Test, and Honey and Mumford's Learning Style are most commonly used by researchers. In this discussion, we will explore these different learning style models and their application in personalized learning.

2.4.1. FSLSM

The Felder-Silverman Learning Style Model [75, 76] is a well-regarded theoretical framework developed by Felder and Silverman. This model not only categorizes students based on how they receive and process information but also aligns teaching styles with specific instructional methods suitable for different groups of students. FSLSM proposes dimensions of learning and teaching styles, matching preferred learning styles with corresponding teaching styles. It characterizes learners along the following dimensions: 1. Active and Reflective: Active learners engage with learning materials actively, enjoy communicating and discussing in groups. Reflective learners, however, prefer to focus on the learning materials alone or with small groups, processing information internally. 2.Sensing vs. Intuitive: Sensing learners follow standard learning materials and procedures to solve problems, showing patience and attention to detail. Intuitive learners, in contrast, are more innovative, preferring abstract learning materials and focusing on the theoretical underpinnings of topics. 3.Visual-Verbal: Visual learners understand and remember concepts through diagrams, videos, and flowcharts. Verbal learners, on the other hand, prefer conversations, written documents, and verbal explanations. 4.Sequential and Global: Sequential learners grasp concepts incrementally, following a logical flow of understanding. Global learners, however, absorb information in a holistic manner, often skipping steps in the conventional sequential process. FSLSM can be effectively applied in web-based personalized e-learning platforms to tailor educational experiences to individual learning preferences. This model enhances the alignment between teaching methods and student learning styles, potentially improving educational outcomes [77].

2.4.2. VARK

The VARK model [78], developed by Neil Fleming in 1987, is a learning style framework designed to understand a learner's characteristics through a set of 16 questions. In online learning, VARK helps cater to different learning preferences through four primary modalities: 1. Visual (V): Learners with a visual preference absorb information best through graphical formats such as diagrams, maps, graphs, flowcharts, and designs. This does not include static pictures of real-world entities. 2. Aural/Auditory (A): Aural learners thrive on spoken information. They learn effectively through lectures, speaking, group discussions, informal talks, chats, and colloquial examples. 3. Read/Write (R): This modality is suited for learners who prefer written documents. They favor text-based materials like reports, lecture notes, assignments, PowerPoint presentations, and textbooks. 4. Kinesthetic (K): Kinesthetic learners excel through hands-on experiences. They prefer learning with concrete representations, simulations, case studies, and other experiential learning methods. VARK's modalities help tailor online learning environments to accommodate diverse learning preferences, enhancing the effectiveness of educational content delivery.

2.4.3. Myers-Briggs Type Indicator test

The Myers-Briggs Type Indicator [79], based on Carl Jung's psychological types, categorizes individuals into four main dimensions: 1. Extraversion (E) or Introversion (I): This dimension describes how individuals interact with the external world. Extraverts are outgoing and energized by social interaction, while introverts are more reserved and gain energy from solitary activities. 2. Sensing (S) or Intuition (N): This dimension focuses on how people perceive and gather information. Sensors rely on concrete, factual information, and details, whereas Intuitives prefer abstract concepts and ideas. 3. Thinking (T) or Feeling (F): This dimension relates to decision-making processes. Thinkers prioritize logic and objective criteria, while feelers prioritize personal values and their impact on others. 4. Judging (J) or Perceiving (P): This dimension describes how individuals approach the external world. Judgers prefer structure, planning, and decisiveness, whereas perceivers are more flexible, spontaneous, and adaptable. MBTI combines these preferences to form 16 distinct personality types, such as ISTP, which stands for Introversion, Sensing, Thinking, and Perceiving. These personality types can provide insights into learning preferences and can be used to tailor educational approaches accordingly.

2.4.4. Mumford learning model

The Mumford Learning Model [74], uses the Learning Style Questionnaire (LSQ) to categorize learners into four distinct groups: 1. Activists: These learners prefer hands-on experiences and learn best by engaging in experiments and projects. 2. Theorists: Theorists focus on understanding theories and hypotheses, requiring specific goals and a structured approach to their learning process. 3. Pragmatists: Pragmatist learners are interested in the practical application of concepts, always thinking about how to implement what they have learned in real-world scenarios. 4. Reflectors: Reflectors take time to observe and consider different aspects of the concepts they encounter, analyzing them from various perspectives before drawing conclusions. This model helps educators tailor their teaching strategies to match the diverse learning preferences of their students, enhancing the overall effectiveness of the educational experience.

3. RESEARCH GAP

The literature under investigation has been thoroughly reviewed. However, several gaps were identified during the review. Researchers proposed various programming language learning strategies and instructional approaches to assist learners in overcoming the hurdles in programming. However little effort and work has been done to address individual learning preferences and in effect to set up a personalized learning environment exclusively for programming learning.

The main research gap found while analyzing the learning style models and instructional approaches are

a. Programming is particularly challenging for novice learners, with studies highlighting high dropout rates in introductory programming courses at universities worldwide [1-3]. Although researchers have proposed various instructional methods, the blended learning approach has shown promising results [60]. Blended learning, a method that combines traditional classroom instruction with e-learning support, has been found to outperform conventional classroom teaching [61-66]. Despite the demonstrated effectiveness of blended learning in enhancing learning outcomes, it is not widely implemented in introductory programming courses. Many institutions still rely on traditional teaching methods, possibly due to limited awareness of blended learning's benefits, resource constraints, or a lack of tailored materials that address the specific challenges of programming. This underutilization represents a significant missed opportunity, as blended learning offers a structured yet flexible environment where foundational programming skills could be taught with personalized support.

b. Individual learner characteristics are crucial in programming instruction. In traditional learning models, however, instructors often assume all students have the same cognitive abilities. It is widely recognized that this adaptive approach can enhance learners' performance and improve the overall quality of the learning experience [55, 56, 58]. Adaptive e-learning is a teaching approach that tailors instructional content to match the unique learning styles or preferences of each student. ALEs are tailored to accommodate learners' unique styles, recognizing that individuals vary in how they engage with and process new information. However, no research was conducted to propose a learning style model exclusively for programming learning. The existing learning style models [74, 75, 78, 79] were designed generally to identify the learner's learning preferences. Moreover, a significant limitation of current learning style models is that they rely on questionnaires to determine learning preferences. Being older frameworks, these models have not evolved to address recent advancements in online and e-learning needs.

c. Instructional video-based e-learning environments have been thoroughly leveraged in the post-COVID time. As a result, Universities and EdTech platforms offering e-learning courses often deliver content through instructional videos, which help learners absorb information more effectively. While there is extensive research [80-90] on the impact of instructional videos on teaching and learning, a key limitation is the lack of attention to individual learner attributes-specifically, learning styles and preferences. Most studies have assumed a uniform learning preference among students, designing instructional videos under the assumption that all learners share a single learning preference and learning style.

Furthermore, a broad range of research methods could be employed to address the difficulty in programming by analyzing how a blended approach helps in assimilating programming instruction, how ALEs work with learning style models, especially for programming, and how personalized instructional videos, tailored to individual learning styles, can accommodate diverse learning paces and cognitive differences.

4. RESEARCH METHODOLOGY

This study emphasizes the careful selection of relevant past research. To examine various programming language learning models, instructional approaches, current trends, and future directions in programming education, a thorough review of relevant studies was conducted. Key articles and papers were selected that provide a comprehensive discussion of these topics, focusing on their significance to the field of programming education. These studies were reviewed in detail to identify insights into effective teaching strategies, challenges, and innovative approaches specific to programming. Citations from these sources have been carefully integrated into the study, ensuring a strong alignment with the objectives of the current research area. This approach provides a solid foundation for understanding the evolution of programming education and highlights areas for potential development in future instructional methodologies.

A thorough literature review has been conducted by examining articles, books, journals, case studies, and conference papers, gathering relevant information and data spanning from 2014 to 2023. The selection process was conducted with careful precision. Several guidelines are typically followed in systematic reviews, particularly in fields like medicine and education. The initial phase, identification, involved searching for research items most relevant to the study's predefined objectives. Next, in the screening phase, a list of potential research items was analyzed for content that precisely aligned with the study's objectives. Finally, the disclosure step involved presenting the final review sample, ensuring transparency by fully listing all research items used. This three-step approach ensured the review sample aligned with key attributes-structure, transparency, and comprehensiveness.

To locate relevant items, databases like Google Scholar and Crossref were employed, with keywords and related terms used to identify, categorize, and filter the studies. Titles, abstracts, and main article content were examined carefully for relevance, resulting in a final selection of approximately 95 studies. These studies were organized into themes including learning environments, strategies, instructional approaches, learning style models, cognitive factors, and current trends in instructional approaches for programming education.

While the review process was extensive, some limitations were noted. This study primarily focused on learning models and instructional approaches within programming education, leaving certain areas less explored. Specifically, a deeper examination of various implementation techniques and a comparative analysis of machine learning and deep learning methods applied to address cutting-edge challenges were not covered. Future studies should consider these aspects to provide a more comprehensive understanding of programming education and enhance the methodological rigor of systematic reviews in this field. This expanded approach would allow for a clearer perspective on emerging technologies and their potential applications in improving programming instruction.

5. TRENDS AND FUTURE DIRECTIONS

Existing educational technologies and pedagogical methods were disrupted during the post-COVID period, and new normal technological advancements replaced them partly or entirely. Blending multiple instructional approaches evolved in online and/or e-learning learning modes. The blended learning approach is notable in the post-COVID period. It harnesses the advantages of classroom teaching and the e-learning mode of instructional delivery. Subsequently, a noteworthy change in the instructional video-based learning platforms also increased exponentially. It gave birth to adaptive environments that deliver instructional material according to the learner's preferences or choices.

5.1. Blended ALEs

Blended learning, a teaching approach that integrates traditional classroom instruction with online and mobile activities to support continuous learning, offers distinct advantages over relying solely on one mode of instructional delivery. This formal education model incorporates digital and online media while granting students some degree of control over aspects such as timing, location, pathway, or pace of learning. In traditional programming courses, which often focus heavily on theory with limited hands-on practice, teaching beginners can be challenging due to the complex nature of programming, varying student characteristics, and standard teaching methods. Implementing a blended learning approach offers a promising solution to overcome these obstacles in programming education. [91].

Blended learning has gained traction among institutions and students seeking to enhance programming education. Research indicates that blended learning surpasses traditional instructional methods in effectiveness and can significantly enrich students' educational experiences [92]. Despite these benefits, a notable gap exists in understanding and implementing blended learning within introductory programming courses considering learners' learning preferences. To address this, integrating Adaptive e-learning with blended teaching is a better pedagogical technique that can be adopted in programming learning. It provides flexibility for learners at different paces when programming is introduced. Adaptive e-learning involves tailoring instructional content to match learners' styles or preferences. These systems create models based on each learner's needs. This adaptive approach is widely recognized for enhancing learners' performance and the overall quality of the learning process [93]. Typically, an adaptive e-learning environment includes three key models: a) the Content model, which outlines the structure of instructional material and learning outcomes; b) the Instructional model, which identifies and delivers personalized instructional content; and c) the Learner model; which tracks learner characteristics and responses to refine the system [77]. continuously.

5.2. Learning style enabled instructional video ALEs

Learning style is essential while designing and delivering learning materials to learners. The diverse nature of human beings is reflected in their adoption of different learning preferences and receptive perspectives. Learning style refers [74] to how each person prefers to learn and process information. It's about understanding the best way for an individual to acquire knowledge and retain it. In e-learning, understanding your learning style can impact how you process and apply information effectively.

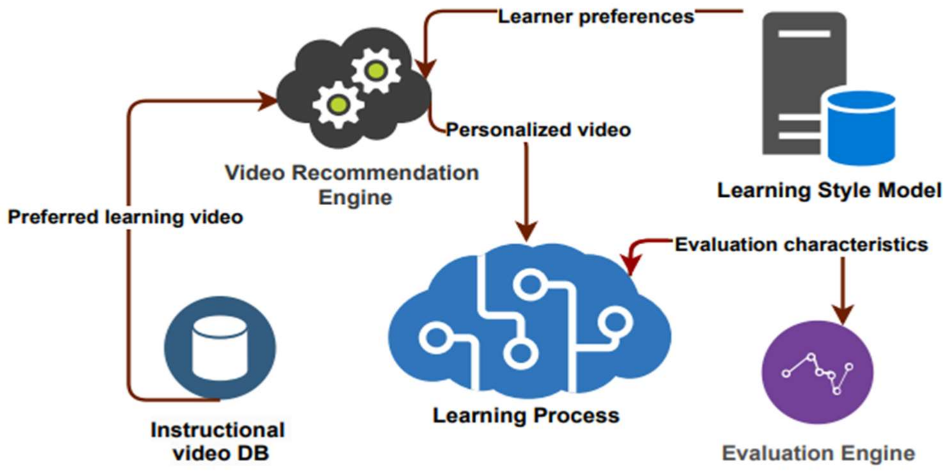

Figure 3: A General Model Of Learning Style Enabled Instructional Video Ales

In conventional adaptive learning environments, the Instructional model delivers personalized learning objects (LOs) that more or less match the learner's preference. LOs are usually presentation slides, lecture notes, or textbook materials for ease of construction. After the COVID-19 pandemic, pedagogical approaches shifted, and conventional LOs gave way to online and pre-recorded instructional videos. According to the cognitive paradigm of multimodal learning, visual and auditory engagement plays a significant role in assimilating educational information. Meta-analyses have demonstrated that integrating complementary information through auditory and visual channels significantly enhances learning outcomes compared to relying on a single channel [94]. As a result, dual-channel processing allows learners to absorb and retain information more effectively, as the brain can handle verbal and visual information simultaneously without overloading one cognitive channel. Therefore, many e-learning environments, particularly those focused on programming education, leverage video content to deliver instructional material [95]. Videos are preferred over traditional text and audio-based approaches because they combine visual elements like code demonstrations and graphical explanations with auditory instructions, creating a richer, more engaging learning experience. This multimodal approach helps learners understand complex programming concepts more thoroughly and retain information longer.

Existing literature [80-90] overlooked a crucial learner attribute: each student's learning style and preference. These studies assumed that all students shared a uniform learning preference while watching and learning from instructional videos. This assumption neglects the diverse ways in which students absorb and process information. Different learners may prefer visual aids, textual explanations, auditory guidance, or a combination of these methods. By not considering these varied preferences, the studies fail to address students' individual needs, potentially limiting the effectiveness of instructional videos for a significant portion of the learning audience.

Integrating individual learning preference instructional videos in an ALE presents a highly effective approach to personalized education [96, 97]. This method acknowledges the diverse ways students process and absorb information, offering a customized educational experience that adjusts to each student's unique preferences (Fig.3). For instance, some students could gain more from visual aids like diagrams and animations. In contrast, others might find textual explanations or auditory instructions more understandable within the video learning mode. By incorporating these varying instructional styles into an adaptive e-learning platform, the system can dynamically adjust the content delivery to match the specific learning preferences of each student.

These integrations enhance learners' engagement and retention rates by providing a more intuitive and accessible learning experience. Instead of a one-size-fits-all approach, students receive content in the format that best suits their cognitive and perceptual strengths, leading to more effective learning outcomes. This personalized approach can also increase motivation and reduce frustration as students interact with instructional materials that resonate with their preferred learning style. Overall, implementing individual learning preference instructional videos within an adaptive e-learning environment can significantly improve the quality of education and student success rates.

6. OPEN RESEARCH ISSUES

Despite significant advancements in programming education, several critical challenges remain unaddressed, presenting numerous open research opportunities. As learners come from diverse backgrounds with varying cognitive abilities, preferences, and motivations, existing instructional methods often fail to provide personalized and effective learning experiences. Key areas requiring further exploration include the integration of adaptive learning technologies, the development of interdisciplinary curricula, the creation of personalized learning materials, and the improvement of collaborative and peer-assisted learning methods. Addressing these open issues is essential to overcome current limitations and improve the overall effectiveness of programming education.

6.1. Adaptive and personalized learning models

AI and machine learning have the potential to revolutionize education by creating truly adaptive learning systems that personalize the learning experience based on the needs, preferences, and progress of individual students. These systems can dynamically adjust content, pace, and feedback to match learners' cognitive abilities and learning styles. Machine learning algorithms can analyze large datasets of student interactions to predict difficulties, recommend appropriate resources, and tailor assessments. Effective frameworks for integrating adaptive systems with traditional teaching should combine human expertise with AI-driven insights. This ensures that educators maintain control over the instructional process. At the same time, they can benefit from real-time analytics and personalized interventions provided by AI. Hybrid models, where AI supports teachers by automating routine tasks such as grading and tracking progress, allow educators to focus on more complex aspects of teaching, such as fostering critical thinking and creativity. In addition, the frameworks should emphasize transparency and interpretability, allowing teachers and students to understand how adaptive recommendations are made, thus fostering trust in the system.

6.2. Human-AI collaboration in programming learning

The optimal balance between automated systems and human educators lies in leveraging the strengths of both to create a more effective and engaging learning environment. Automated systems excel at providing personalized content, providing instant feedback, and analyzing large amounts of student data to identify learning patterns and potential difficulties. They can handle routine tasks such as grade and progress tracking, allowing educators to focus on higher-order skills such as critical thinking, creativity, and emotional support. However, human educators play a crucial role in motivating learners, fostering a sense of community, and adapting to nuanced social and emotional cues that machines cannot fully interpret. Achieving this balance involves integrating AI as a supportive tool rather than a replacement, ensuring that automation enhances, rather than diminishes, the educator's role. Collaborative frameworks should also provide educators with intuitive interfaces to interpret AI-driven insights and allow them to make informed decisions, maintaining a human-centered approach to learning. Learning Classifier Systems (LCS) can be utilized to generate dynamic rules to achieve the optimal balance, human-centered informed decisions between automated systems and human educators.

6.3. Learner specific programming learning video creation

Creating learner-specific programming learning videos tailored to individual preferences poses significant challenges for educators, making it an open research question. Personalizing video content requires adapting factors such as teaching style, personalized examples, and difficulty level to align with each learner's cognitive style, prior knowledge, and learning speed. Producing such customized videos manually from scratch is time-consuming and resource-intensive. This highlights the need for automated or semi-automated systems that dynamically generate or adapt video content based on learner profiles. Future research could focus on developing AI-driven frameworks that use natural language processing, video synthesis, and machine learning to create personalized instructional videos. Such systems could enable educators to input high-level content, which the AI would transform into tailored videos. Finally, addressing the learners' specific needs while significantly reducing the workload on educators.

7. LIMITATIONS

This section presents a few limitations that need to be considered for contextualizing its findings. First, the scope of the review is limited to programming learning, which narrows its applicability to other fields of education. While programming education has its unique set of challenges and instructional needs, the findings and proposed approaches may not directly translate to other disciplines, such as mathematics or language learning. Future research should explore how the proposed methods might be adapted or expanded for broader educational contexts. Second, despite being generally acknowledged in the field of education, learning styles could not always precisely represent the actual preferences or aptitudes of certain students. Learning styles alone are insufficient for predicting how a student learns best, as they can oversimplify complex cognitive processes. As a result, while this study identifies learning styles as one of the potential path for improving programming education, their usefulness in developing personalised learning experiences may be limited if they are used without accompanying cognitive or behavioural data. Third, the availability of research that integrates learning styles specifically with programming education is relatively limited. This lack of extensive prior work constrains the review's ability to draw robust conclusions and propose well-supported frameworks. The study, therefore, mainly compiles existing, isolated works rather than synthesizing well-established models. Further research is needed to build a comprehensive understanding of how learning styles can be effectively leveraged in programming pedagogy. Lastly, the nature of this review is primarily a plain literature survey, aimed at consolidating existing research rather than providing a detailed critical analysis. While this approach helps to map the current landscape of programming education and identify research gaps, it does not delve deeply into evaluating the strengths and weaknesses of specific methods. A more critical evaluation, which could assess the comparative effectiveness of different approaches and offer in-depth critiques, would provide greater value for guiding future research efforts. Addressing these limitations in future studies could enhance the overall impact of research on programming education by offering more comprehensive insights and practical solutions.

8. CONCLUSION

This review examines recent advancements in programming learning models and instructional approaches, providing an in-depth analysis of programming learning environments, including text-based, block-based, game-based, and visual programming formats. A thorough evaluation of instructional methods and learning style models has also been conducted, highlighting the importance of integrating various approaches to enhance learner engagement, retention, and overall learning experiences. Our findings suggest that combining adaptive video learning environments with individualized learning styles can significantly improve online and e-learning formats for programming education. Future work could extend this review to explore specific techniques, such as the learning style model for programming and personalized instructional video models, which promise to improve programming instruction and learner outcomes.

REFERENCES

[1]. Mutanga MB. The Effect of Cognitive Factors in Determining students' Success in Computer Programming. Journal of Theoretical and Applied Information Technology. 2020;98(17):3606–3618.

[2]. Weintrop D, Wilensky U. How block-based, text-based, and hybrid block/text modalities shape novice programming practices. International Journal of Child-Computer Interaction. 2018 Sep;17:83–92. https://doi.org/10. 1016/j.ijcci.2018.04.005.

[3]. Repenning A. Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets. Journal of Visual Languages and Sentient Systems. 2017 Jul;3(1):68–91. https://doi.org/10.18293/vlss2017-010.

[4]. Repenning A, Sumner T. Agentsheets: a medium for creating domain-oriented visual languages. Computer. 1995 Mar;28(3):17–25. https://doi.org/10.1109/2.366152.

[5]. Wong YS, Hayati MYM, Tan WH. A propriety game-based learning game as learning tool to learn object-oriented programming paradigm. In: Serious Games: Second Joint International Conference, JCSG 2016, Brisbane, QLD, Australia, September 26-27, 2016, Proceedings 2. Springer; 2016. p. 42–54.

[6]. Asad K, Tibi M, Raiyn J. Primary School Pupils' Attitudes toward Learning Programming through Visual Interactive Environments. World Journal of Education. 2016 Oct;6(5). https://doi.org/10.5430/wje.v6n5p20.

[7]. Weintrop D. Block-based programming in computer science education. Communications of the ACM. 2019 Jul;62(8):22–25. https://doi.org/10.1145/3341221.

[8]. Malik SI, Al-Emran M, Mathew R, Tawafak RM, Alfarsi G. Comparison of E-Learning, M-Learning and Game-based Learning in Programming Education – A Gendered Analysis. International Journal of Emerging Technologies in Learning (iJET). 2020 Aug;15(15):133. https://doi.org/10.3991/ijet.v15i15.14503.

[9]. Mathrani A, Christian S, Ponder-Sutton A. PlayIT: Game based learning approach for teaching programming concepts. Journal of Educational Technology & Society. 2016;19(2):5–17.

[10]. M P, N NM, Dakshina R, S S, R BS. Learning Analytics: Game-based Learning for Programming Course in Higher Education. Procedia Computer Science. 2020;172:468–472. https://doi.org/10.1016/j.procs.2020.05.143.

[11]. Rugelj J, Lapina M. Game design based learning of programming. In: Proceedings of SLET-2019–International Scientific Conference Innovative Approaches to the Application of Digital Technologies in Education and Research; 2019. p. 20–23.

[12]. Chang CS, Chung CH, Chang JA. Influence of problem-based learning games on effective computer programming learning in higher education. Educational Technology Research and Development. 2020 Jun;68(5):2615–2634. https://doi.org/10. 1007/s11423-020-09784-3.

[13]. Holenko Dlab M, Hoic-Bozic N. Effectiveness of game development-based learning for acquiring programming skills in lower secondary education in Croatia. Education and Information Technologies. 2021 Mar;26(4):4433–4456. https://doi.org/10. 1007/s10639-021-10471-w.

[14]. Hu Y, Su CY, Fu A. Factors influencing younger adolescents' intention to use gamebased programming learning: A multigroup analysis. Education and Information Technologies. 2022 Mar;27(6):8203–8233. https: //doi.org/10.1007/s10639-022-10973-1.

[15]. Tsai CY. Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. Computers in Human Behavior. 2019 Jun;95:224–232. https://doi.org/10. 1016/j.chb.2018.11.038.

[16]. Hu Y, Chen CH, Su CY. Exploring the Effectiveness and Moderators of Block-Based Visual Programming on Student Learning: A Meta-Analysis. Journal of Educational Computing Research. 2020 Jul;58(8):1467–1493. https://doi.org/10.1177/0735633120945935.

[17]. Batiha Q, Sahari N, Aini N, Mohd N. Adoption of Visual Programming Environments in Programming Learning. International Journal on Advanced Science, Engineering and Information Technology. 2022 Sep;12(5):1921. https://doi.org/10.18517/ ijaseit.12.5.15500.

[18]. Iskrenovic-Momcilovic O. Choice of visual programming language for learning programming. International Journal of Computers. 2017;2.

[19]. Kesler A, Shamir-Inbal T, Blau I. Active Learning by Visual Programming: Pedagogical Perspectives of Instructivist and Constructivist Code Teachers and Their Implications on Actual Teaching Strategies and Students' Programming Artifacts. Journal of Educational Computing Research. 2021 Jun;60(1):28–55. https://doi.org/10.1177/ 07356331211017793.

[20]. Dina NZ, Wuryanto E, Marjianto RS. Evaluation on the Effectiveness of Visual Learning Environment on Programming Course From Students' Perspectives. IIUM Engineering Journal. 2019 Jun;20(1):100–107. https:// doi.org/10.31436/iiumej.v20i1.993.

[21]. Han KW, Lee E, Lee Y. The Impact of a PeerLearning Agent Based on Pair Programming in a Programming Course. IEEE Transactions on Education. 2010 May;53(2):318–327. https://doi.org/10.1109/te.2009.2019121.

[22]. Ngatirin NR, Zainol Z, Fang WC. Mobile Agile Peer Assisted Learning. In: 2016 3rd International Conference on Computer and Information Sciences (ICCOINS). IEEE; 2016. Available from: http://dx.doi.org/10. 1109/iccoins.2016.7783191.

[23]. Mart´ın-Ramos P, Lopes MJ, Lima da Silva MM, Gomes PEB, Pereira da Silva PS, Domingues JPP, et al. First exposure to Arduino through peer-coaching: Impact on students' attitudes towards programming. Computers in Human Behavior. 2017 Nov;76:51–58. https://doi.org/10.1016/j.chb. 2017.07.007.

[24]. Altintas T, Gunes A, Sayan H. A peerassisted learning experience in computer programming language learning and developing computer programming skills. Innovations in Education and Teaching International. 2014 Dec;53(3):329–337. https://doi.org/10.1080/ 14703297.2014.993418.

[25]. Webb NM, Ender P, Lewis S. ProblemSolving Strategies and Group Processes in Small Groups Learning Computer Programming. American Educational Research Journal. 1986;23(2):243. https://doi.org/10.2307/ 1162957.

[26]. Wang XM, Hwang GJ. A problem posingbased practicing strategy for facilitating students' computer programming skills in the team-based learning mode. Educational Technology Research and Development. 2017 Nov;65(6):1655–1671. https://doi.org/10. 1007/s11423-017-9551-0.

[27]. Cabrera I, Villalon J, Chavez J. Blending Communities and Team-Based Learning in a Programming Course. IEEE Transactions on Education. 2017 Nov;60(4):288–295. https: //doi.org/10.1109/te.2017.2698467.

[28]. Hsu TC, Abelson H, Patton E, Chen SC, Chang HN. Self-efficacy and behavior patterns of learners using a real-time collaboration system developed for group programming. International Journal of Computer-Supported Collaborative Learning. 2021 Dec;16(4):559–582. https://doi. org/10.1007/s11412-021-09357-3.

[29]. Irwan I, Susanti W, Desnelita Y, Gustientiedina G, Wongso F, Fudholi A. Problem-based Collaborative Learning Strategy in Computer Programming. SAR

Journal - Science and Research. 2022 Mar;p. 40–45. https://doi. org/10.18421/sar51-05.

[30]. Chorfi A, Hedjazi D, Aouag S, Boubiche D. Problem-based collaborative learning groupware to improve computer programming skills. Behaviour & Information Technology. 2020 Jul;41(1):139–158. https: //doi.org/10.1080/0144929x.2020.1795263.

[31]. Bowman NA, Jarratt L, Culver KC, Segre AM. How Prior Programming Experience Affects Students' Pair Programming Experiences and Outcomes. In: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education. ITiCSE '19. ACM; 2019. Available from: http://dx.doi.org/10.1145/3304221.3319781.

[32]. Wei X, Lin L, Meng N, Tan W, Kong SC, Kinshuk. The effectiveness of partial pair programming on elementary school students' Computational Thinking skills and self-efficacy. Computers & Education. 2021 Jan;160:104023. https://doi.org/10. 1016/j.compedu.2020.104023.

[33]. Liu Y, Tong Y, Yang Y. The application of mind mapping into college computer programming teaching. Procedia Computer Science. 2018;129:66–70.

[34]. Zhao L, Liu X, Wang C, Su YS. Effect of different mind mapping approaches on primary school students' computational thinking skills during visual programming learning. Computers & Education. 2022 May;181:104445. https://doi.org/10.1016/j. compedu.2022.104445.

[35]. Su YS, Shao M, Zhao L. Effect of Mind Mapping on Creative Thinking of Children in Scratch Visual Programming Education. Journal of Educational Computing Research. 2021 Dec;60(4):906–929. https://doi.org/10. 1177/07356331211053383.

[36]. Ismail MN, Ngah NA, Umar IN. The Effects of Mind Mapping with Cooperative Learning on Programming Performance, Problem Solving Skill and Metacognitive Knowledge among Computer Science Students. Journal of Educational Computing Research. 2010 Jan;42(1):35–61. https://doi.org/10.2190/ec. 42.1.b.

[37]. Gul S, Asif M, Nawaz Z, Haris Aziz M, Khurram S, Qaiser Saleem M, et al. Sustainable Learning of Computer Programming Languages Using Mind Mapping. Intelligent Automation & Soft Computing. 2023;36(2):1687–1697. https://doi.org/10. 32604/iasc.2023.032494.

[38]. Nathoo A, Gangabissoon T, Bekaroo G. Exploring the Use of Tangible User Interfaces for Teaching Basic Java Programming Concepts: A Usability Study. In: 2019 Conference on Next Generation Computing Applications (NextComp). IEEE; 2019. Available from: http://dx.doi.org/10.1109/ nextcomp.2019.8883563.

[39]. Caceres PC, Venero RP, Cordova FC. Tangible programming mechatronic interface for basic induction in programming. In: 2018 IEEE Global Engineering Education Conference (EDUCON). IEEE; 2018. Available from: http://dx.doi.org/10.1109/ educon.2018.8363226.

[40]. Vidarte JDT, Rinderknecht C, Kim JI, Kim H. A Tangible Interface for Learning Recursion and Functional Programming. In: 2010 International Symposium on Ubiquitous Virtual Reality. IEEE; 2010. Available from: http://dx.doi.org/10.1109/isuvr.2010.18.

[41]. Chis AE, Moldovan AN, Murphy L, Pathak P, Muntean CH. Investigating flipped classroom and problem-based learning in a programming module for computing conversion course. Journal of Educational Technology & Society. 2018;21(4):232–247.

[42]. Ta¸spolat A, Ozdamli F, Soykan E. Pro- ¨ gramming Language Training With the Flipped Classroom Model. SAGE Open. 2021 Apr;11(2):215824402110214. https:// doi.org/10.1177/21582440211021403.

[43]. Ozyurt H, ¨ Ozyurt ¨ O. Analyzing the effects of ¨ adapted flipped classroom approach on computer programming success, attitude toward programming, and programming self-efficacy. Computer Applications in Engineering Education. 2018;26(6):2036–2046.

[44]. C¸ akıro˘glu U, ¨ Ozt¨urk M. Flipped class- ¨ room with problem based activities: Exploring self-regulated learning in a programming language course. JSTOR; 2017. .

[45]. Yan OS, Cheng G. Exploring the impact of flipped classroom on students' acceptance of programming in secondary education. In: 2017 IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE). IEEE; 2017. Available

from: http://dx.doi.org/10.1109/ tale.2017.8252341.
[46]. Ozyurt ¨ O, ¨ Ozyurt H. A qualitative study ¨ about enriching programming and algorithm teaching with flipped classroom approach. Pegem Egitim ve Ogretim Dergisi. 2017;7(2).
[47]. Pattanaphanchai J. An Investigation of Students' Learning Achievement and Perception using Flipped Classroom in an Introductory Programming course: A Case Study of Thailand Higher Education. Journal of University Teaching and Learning Practice. 2019 Dec;16(5):36–53. https://doi.org/10.53761/ 1.16.5.4.
[48]. Law KMY, Lee VCS, Yu YT. Learning motivation in e-learning facilitated computer programming courses. Computers & Education. 2010 Aug;55(1):218–228. https: //doi.org/10.1016/j.compedu.2010.01.007.
[49]. Zın AM, Idrıs S, Subramanıam NK. Improving learning of programming through elearning by using asynchronous virtual pair programming. Turkish Online Journal of Distance Education. 2006;7(3):162–173.
[50]. Chookaew S, Panjaburee P, Wanichsan D, Laosinchai P. A Personalized E-Learning Environment to Promote Student's Conceptual Learning on Basic Computer Programming. Procedia - Social and Behavioral Sciences. 2014 Feb;116:815–819. https://doi. org/10.1016/j.sbspro.2014.01.303.
[51]. Malik SI, Mathew R, Al-Nuaimi R, Al-Sideiri A, Coldwell-Neilson J. Learning problem solving skills: Comparison of E-learning and M-learning in an introductory programming course. Education and Information Technologies. 2019 Mar;24(5):2779–2796. https: //doi.org/10.1007/s10639-019-09896-1.
[52]. Tundjungsari V. E-learning model for teaching programming language for secondary school students in Indonesia. In: 2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV). IEEE; 2016. Available from: http://dx.doi. org/10.1109/rev.2016.7444477.
[53]. Mutiawani V, Juwita. Developing e-learning application specifically designed for learning introductory programming. In: 2014 International Conference on Information Technology Systems and Innovation (ICITSI). IEEE; 2014. Available from: http://dx.doi.org/10. 1109/icitsi.2014.7048250.
[54]. Nariman D. In: Impact of the Interactive e-Learning Instructions on Effectiveness of a Programming Course. Springer International Publishing; 2020. p. 588–597. Available from: http://dx.doi.org/10.1007/ 978-3-030-50454-0 61.
[55]. Troussas C, Krouska A, Sgouropoulou C. A Novel Teaching Strategy Through Adaptive Learning Activities for Computer Programming. IEEE Transactions on Education. 2021 May;64(2):103–109. https://doi.org/10. 1109/te.2020.3012744.
[56]. Gavrilovi´c N, Arsi´c A, Domazet D, Mishra A. Algorithm for adaptive learning process and improving learners' skills in Java programming language. Computer Applications in Engineering Education. 2018 Aug;26(5):1362–1382. https://doi.org/10. 1002/cae.22043.
[57]. Chrysafiadi K, Virvou M. Fuzzy Logic for Adaptive Instruction in an E-learning Environment for Computer Programming. IEEE Transactions on Fuzzy Systems. 2015 Feb;23(1):164–177. https://doi.org/10.1109/ tfuzz.2014.2310242.
[58]. Chrysafiadi K, Virvou M, Sakkopoulos E. In: Optimizing Programming Language Learning Through Student Modeling in an Adaptive Web-Based Educational Environment. Springer International Publishing; 2019. p. 205–223. Available from: http://dx.doi.org/ 10.1007/978-3-030-13743-4 11.
[59]. Chrysafiadi K, Virvou M, Tsihrintzis GA, Hatzilygeroudis I. An Adaptive Learning Environment for Programming Based on Fuzzy Logic and Machine Learning. International Journal on Artificial Intelligence Tools. 2023 Aug;32(05). https://doi.org/10.1142/ s0218213023600114.
[60]. Luo J, Wang T. Analyzing Students' Behavior in Blended Learning Environment for Programming Education. In: Proceedings of the 2020 The 2nd World Symposium on Software Engineering. WSSE 2020. ACM; 2020. Available from: http://dx.doi.org/10. 1145/3425329.3425346.
[61]. Tadlaoui MA, Chekou M. A blended learning approach for teaching python programming language: towards a post pandemic pedagogy. International Journal of Advanced Computer Research. 2021

Jan;11(52):13–22. https://doi.org/10.19101/ijacr.2020.1048120.

[62]. Mabni Z, Shamsudin N, Aliman S, Abdul Latif R. Factors Influencing Students' Performance in the First Computer Programming Course Taught Using Blended Learning Approach. Environment-Behaviour Proceedings Journal. 2020 Dec;5(SI3):181–186. https://doi.org/10.21834/ebpj.v5isi3.2559.

[63]. Srivatanakul T. Emerging from the pandemic: instructor reflections and students' perceptions on an introductory programming course in blended learning. Education and Information Technologies. 2022 Nov;28(5):5673–5695. https://doi.org/10.1007/s10639-022-11328-6.

[64]. Murai Y, Muramatsu H. Application of creative learning principles within blended teacher professional development on integration of computer programming education into elementary and middle school classrooms. Information and Learning Sciences. 2020 Jun;121(7/8):665–675. https://doi.org/ 10.1108/ils-04-2020-0122.

[65]. F¨orster A, Dede J, Udugama A, F¨orster A, Helms D, Kniefs L, et al. A blended learning approach for an introductory computer science course. Education Sciences. 2021;11(8):372.

[66]. Ya˘gci M. A Web-based Blended Learning Environment for Programming Languages: Students' Opinions. Journal of Education and Training Studies. 2017 Feb;5(3):211. https://doi.org/10.11114/jets.v5i3.2118.

[67]. Hwang GJ, Sung HY, Chang SC, Huang XC. A fuzzy expert system-based adaptive learning approach to improving students' learning performances by considering affective and cognitive factors. Computers and Education: Artificial Intelligence. 2020;1:100003. https://doi.org/10.1016/j.caeai.2020.100003.

[68]. Msane J, Mutanga B, Chani T. Students' Perception of the Effect of Cognitive Factors in Determining Success in Computer Programming: A Case Study. International Journal of Advanced Computer Science and Applications. 2020;11(7). https://doi.org/10.14569/ijacsa.2020.0110724.

[69]. Dirzyte A, Vijaikis A, Perminas A, Rimasiute-Knabikiene R, Kaminskis L, Zebrauskas G. Computer Programming ELearners' Personality Traits, Self-Reported Cognitive Abilities, and Learning Motivating Factors. Brain Sciences. 2021 Sep;11(9):1205. https://doi.org/10.3390/brainsci11091205.

[70]. Talsma K, Sch¨uz B, Norris K. Miscalibration of self-efficacy and academic performance: Self-efficacy ≠ self-fulfilling prophecy. Learning and Individual Differences. 2019 Jan;69:182–195. https://doi.org/10.1016/j. lindif.2018.11.002.

[71]. Kanaparan G, Cullen R, Mason D. Effect of Self-efficacy and Emotional Engagement on Introductory Programming Students. Australasian Journal of Information Systems. 2019 Jul;23. https://doi.org/10.3127/ajis.v23i0.1825.

[72]. Ismayilova K, Klassen RM. Research and teaching self-efficacy of university faculty: Relations with job satisfaction. International Journal of Educational Research. 2019;98:55–66. https://doi.org/10.1016/j.ijer.2019.08.012.

[73]. Aivaloglou E, Hermans F. Early Programming Education and Career Orientation: The Effects of Gender, Self-Efficacy, Motivation and Stereotypes. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education. SIGCSE '19. ACM; 2019. Available from: http://dx.doi.org/10. 1145/3287324.3287358.

[74]. Honey P, Mumford A, et al. The manual of learning styles. vol. 3. Peter Honey Maidenhead; 1992.

[75]. Felder RM, Silverman LK, et al. Learning and teaching styles in engineering education. Engineering education. 1988;78(7):674–681.

[76]. Chen SY, Wang JH. Individual differences and personalized learning: a review and appraisal. Universal Access in the Information Society. 2021;20(4):833–849.

[77]. Ali N, Eassa F, Hamed E. Personalized learning style for adaptive e-learning system. International Journal of Advanced Trends in Computer Science and Engineering. 2019;8(1):223–230. https://doi.org/10.30534/ijatcse/2019/4181.12019.

[78]. Fleming N, Mills C. VARK: A guide to learning styles. 2001;.

[79]. El Bachari E, Abdelwahed E, El Adnani M. Design of an adaptive e-learning model based on learner's personality. Ubiquitous Computing and Communication Journal. 2010;5(3):1–8.

[80]. Noetel M, Griffith S, Delaney O, Sanders T, Parker P, del Pozo Cruz B, et al. Video improves learning in higher education: A

systematic review. Review of educational research. 2021;91(2):204–236. https://doi. org/10.31234/osf.io/kynez.
[81]. Sabli´c M, Mirosavljevi´c A, Skugor A. Video- ˇ based learning (VBL)—past, present and future: An overview of the research published from 2008 to 2019. Technology, Knowledge and Learning. 2021;26(4):1061–1077. https: //doi.org/10.1007/s10758-020-09455-5.
[82]. Hsu FH, Lin IH, Yeh HC, Chen NS. Effect of Socratic Reflection Prompts via video-based learning system on elementary school students' critical thinking skills. Computers & Education. 2022;183:104497. https://doi. org/10.1016/j.compedu.2022.104497.
[83]. Richter E, Hußner I, Huang Y, Richter D, Lazarides R. Video-based reflection in teacher education: Comparing virtual reality and real classroom videos. Computers & Education. 2022;190:104601. https://doi.org/10. 1016/j.compedu.2022.104601.
[84]. Kilbury M, B¨ohnke A, Thiel F. Producing Staged Videos for Teacher Education: Development and Content Validation of Video Scripts on the Topic of Handling Classroom Disruptions. Education Sciences. 2023;13(1):56. https://doi.org/10. 3390/educsci13010056.
[85]. Lin YT, Yeh MKC, Tan SR. Teaching programming by revealing thinking process: Watching experts' live coding videos with reflection annotations. IEEE Transactions on Education. 2022;65(4):617–627. https://doi. org/10.1109/te.2022.3155884.
[86]. Breslyn W, Green AE. Learning science with YouTube videos and the impacts of Covid19. Disciplinary and interdisciplinary science education research. 2022;4(1):1–20. https:// doi.org/10.1186/s43031-022-00051-4.
[87]. Scagnoli NI, Choo J, Tian J. Students' insights on the use of video lectures in online classes. British Journal of Educational Technology. 2019;50(1):399–414.
[88]. Liao CW, Chen CH, Shih SJ. The interactivity of video and collaboration for learning achievement, intrinsic motivation, cognitive load, and behavior patterns in a digital gamebased learning environment. Computers & Education. 2019;133:43–55. https://doi.org/ 10.1016/j.compedu.2019.01.013.
[89]. Wachtler J, Hubmann M, Z¨ohrer H, Ebner M. An analysis of the use and effect of questions in interactive learning-videos. Smart Learning Environments. 2016;3(1):1–16. https:// doi.org/10.1186/s40561-016-0033-3.
[90]. Kleftodimos A, Evangelidis G. Using open source technologies and open internet resources for building an interactive video based learning environment that supports learning analytics. Smart Learning Environments. 2016;3:1–23. https://doi.org/10.1186/ s40561-016-0032-4.
[91]. Demaidi MN, Qamhieh M, Afeefi A. Applying blended learning in programming courses. IEEE Access. 2019;7:156824–156833. https: //doi.org/10.1109/access.2019.2949927.
[92]. Alammary A. Blended learning models for introductory programming courses: A systematic review. PloS one. 2019;14(9):e0221765. https: //doi.org/10.1371/journal.pone.0221765.
[93]. Kolekar SV, Pai RM, MM MP. Prediction of Learner's Profile Based on Learning Styles in Adaptive E-learning System. International Journal of Emerging Technologies in Learning. 2017;12(6). https://doi.org/10.3991/ijet. v12i06.6579.
[94]. Mayer RE. Applying the science of learning: evidence-based principles for the design of multimedia instruction. American psychologist. 2008;63(8):760. https://doi.org/10. 1037/0003-066x.63.8.760.
[95]. Sanal Kumar TS, Thandeeswaran R. A general model for an instructional video-based personalized programming learning environment and its practical implications. In: 2023 12th International Conference on Advanced Computing (ICoAC); 2023. p. 1–6.
[96]. T S SK, Thandeeswaran R. Adapting videobased programming instruction: An empirical study using a decision tree learning model. Education and Information Technologies. 2024 Jan;https://doi.org/10.1007/ s10639-023-12390-4. [97] Sanal Kumar TS, Thandeeswaran R. An improved adaptive personalization model for instructional video-based e-learning environments. Journal of Computers in Education. 2024 Jan;https://doi.org/10.1007/ s40692-023-00310-x.

Journal of Theoretical and Applied Information Technology
31st May 2025. Vol.103. No.10
© Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ABBREVIATIONS

The following abbreviations are shown in Table.1 are used in this manuscript.

Table 1: Abbreviations

Abbreviations	Meaning
BI	Behavioral Intention
CBS-MM	Construct By Self Mind Mapping
CL	Cooperative Learning
CoL	Community of inquiry learning framework
COS-MM	Construct-On Scaffold Mind Mapping
CS	Computer Science
CT	Computational Thinking
DP	Dynamic Programming
FC	Flipped Classroom
FSLSM	Felder Silverman Learning Style Model
FYO	Follow Your Objective
GBL	Game-Based Learning
GUI	Graphical User Interface
IDE	Integrated Development Environment
ITS	Intelligent Tutoring System
LO	Learning Object
MAI	MIT App Inventor
MBTI	Myers-Briggs Type Indicator
MIT	Massachusetts Institute of Technology
MMCL	Mind Mapping Combined with Cooperative Learning
PAL	Peer Assisted Learning
PASS	Programming Assignment aSsessment System
PBL	Problem-Based Learning
RBT	Revised Bloom Taxonomy
TBL	Team-Based Learning
TUI	Tangible User Interface
VARK	Visual, Auditory, Read/Write, and Kinesthetic
VPL	Visual Programming Language
VPP	Virtual Pair Programming