Computers and Education: Artificial Intelligence 8 (2025) 100403

Contents lists available at ScienceDirect

Computers and Education: Artificial Intelligence

journal homepage: www.sciencedirect.com/journal/computers-and-education-artificial-intelligence

Artificial intelligence in computer programming education: A systematic literature review

Pisut Manorat, Suppawong Tuarob, Siripen Pongpaichet *

Faculty of Information and Communication Technology, Mahidol University, Nakhon Pathom, Thailand

A R T I C L E I N F O

Keywords:
Artificial intelligence
Machine learning
Computer programming education
Systematic literature review

A B S T R A C T

The demand for skilled programmers and the increasing complexity of coding skills have led to a rise in the adoption of artificial intelligence (AI) and machine learning (ML) technologies in computer programming education. Previous research has explored the potential of AI in aspects such as grading assignments, generating feedback, detecting plagiarism, and identifying at-risk students, but there is a lack of systematic reviews focused on AI-powered teaching processes in computer programming classes. To provide a more comprehensive understanding of AI and ML's role in computer programming education, this systematic review examines a wider range of applications across the entire pedagogical process. Analyzing 119 relevant research papers published between 2012 and 2024, this review offers an overview of AI and ML tools and techniques used in various educational contexts. Aligned with instructional design models, the reviewed literature is categorized into four key areas: course design, classroom implementation, assessment and feedback, and performance monitoring. This systematic review not only highlights the practical tools available to instructors but also identifies research trends and potential areas for future exploration in the field of computer programming education.

1. Introduction

While computer programming is one of the foundational skills in several disciplines, many learners find the subject challenging. Challenges faced in introductory programming courses come from several angles: nature of programming language, students, and instructors (Kadar et al., 2021). While the standardization of curriculum and adoption of programming languages with simple syntax can facilitate the process of acquiring programming knowledge, appropriate design of assessment rubrics for novice programmers also plays a crucial role in ensuring that students not only produce functionally correct code but also adhere to appropriate code conventions (Mehmood et al., 2020).

Artificial Intelligence (AI) refers to the reproduction of human behaviors and human intelligence, such as perceiving, learning, and predicting, in a system or a machine (Xu et al., 2021). The education sector has been exploring the integration of AI technologies in performing administrative tasks, assisting instructional duties, and enhancing learning (Chen et al., 2020). Machine Learning (ML), a subfield of AI, inherently differs from rule-based AI that requires explicit programming. By developing appropriate statistical models and algorithms, ML allows machines to learn from data and provide predictions or outcomes for new data points using the knowledge from historical data. Specifically in the field of education research, Educational Data Mining (EDM) and Learning Analytics (LA) have exhibited significant progress from AI and ML technologies. EDM focuses on the investigation and exploration of educational data to create an understanding of students and their learning environment (Baker & Yacef, 2009). Although LA also employs similar techniques used in EDM, it is more learner-focused as it aims to improve learning through data discovery (Clow, 2013). As Deep Learning (DL) has gained more attention due to advancement in computational capacity, such a technology has been utilized in recent EDM in several tasks such as natural language processing and image recognition (Hernández-Blanco et al., 2019). Similarly, Kurilovas (2019) has proposed an ML-based approach to personalized learning in virtual learning environments.

Considerable research has been conducted regarding the applications of AI and ML in teaching computer programming courses in universities. Examples of areas where ML can be implemented in the teaching of computer programming courses are assignment grading (Srikant & Aggarwal, 2014; Rezende Souza et al., 2019; Uchiyama et al., 2014; Takhar & Aggarwal, 2019; Singh et al., 2016), feedback generation (Beck et al., 2018; Wang, 2023; Hoq et al., 2024), plagiarism detection

* Corresponding author.
E-mail addresses: pisut.man@student.mahidol.edu (P. Manorat), suppawong.tua@mahidol.edu (S. Tuarob), siripen.pon@mahidol.edu (S. Pongpaichet).

https://doi.org/10.1016/j.caeai.2025.100403
Received 6 September 2024; Received in revised form 18 February 2025; Accepted 31 March 2025
Available online 2 April 2025

(Hoq et al., 2024; Yasaswi et al., 2017), and at-risk student identification (Jamjoom et al., 2021; Liao et al., 2019; Skalka & Drlik, 2020; Veerasamy et al., 2021a). With a high number of students enrolled in computer programming courses, manual assessment of source code becomes a major workload for instructors and teaching assistants. This situation also stems from a lack of personalized and meaningful feedback to students, which can discourage them from revising their understanding of programming concepts. Some students may resort to submitting plagiarized work instead of learning to solve the problem on their own, further hindering their knowledge and skills. As a result, early identification of low-performing students can help instructors provide additional support to students in need before the end of the semester. Several ML-based solutions have been studied and proposed to tackle these issues in university computer programming courses.

Although there were past review papers on similar topics, they focused on particular tasks in the teaching process. For instance, Kuruppu et al. (2021), Tharmaseelan et al. (2021), and Combéfis (2022) explored the advancement of automated source code assessment in computer programming courses to streamline grading assignment tasks which require a significant amount of time and effort from instructors. Another area where AI and ML were utilized is feedback generation. Feedback plays an integral part in students' learning journey as it identifies how students can improve from their mistakes. While these tools leverage both dynamic assessment and static assessment, the generated feedback is not diverse, restricting the way students solve the problems (Keuning et al., 2016). Besides the limited scope of the past studies, the narrative of the past reviews heavily focused on researchers' perspectives, placing less emphasis on the views of instructors. Notably, previous studies have not adequately highlighted the potential benefits of AI and ML techniques in the classroom — a crucial consideration for instructors deciding whether to adopt these technologies. While the contributions of these reviews were significant in the field, an end-to-end understanding of the utilization of AI and ML in computer programming courses can provide a complete perspective for not only instructors who aim to apply AI technologies in their classrooms but also researchers who intend to broaden the capacity of AI and ML in computer programming education research. Hence, this review will explore the current frontier of AI and ML in computer programming education, aiming to provide valuable insights for both researchers and instructors.

The objective of this systematic literature review is to consolidate recent research appertaining the utilization of AI and ML technologies in enhancing the teaching process of computer programming courses in higher education through the standard systematic literature review methodology. This systematic review aims to examine how computer programming education research has been utilizing AI and ML in their literature, covering the entire teaching process from constructing materials to monitoring learners' performance. This review is structured as follows. Section 2 describes the review methodology, including research questions, search string specifications, and inclusion and exclusion criteria. Section 3 explores the overall demographic of the chosen articles. Section 4 presents a comparative discussion on selected papers in different teaching processes. Section 5 addresses the research questions through analysis derived from the systematic review. Lastly, Section 6 summarizes this systematic review article and provides directions for future research.

2. Review methodology

This systematic literature review follows the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) methods (Page et al., 2021). The process starts with defining the research questions. Next, a search strategy is developed, which includes selecting databases, identifying search keywords, and filtering the papers that match these keywords using specific inclusion and exclusion criteria. The final step involves analyzing all selected articles based on their objectives, methodologies, strengths, and limitations.

Based on the objective of this study, four research questions are raised:

RQ1. Which areas of AI and ML can be used to augment the teaching and learning process in computer programming courses in higher education?
RQ2. What applications do AI and ML take in each teaching process of computer programming courses in higher education?
RQ3. Which AI and ML techniques were used in teaching computer programming courses in higher education?
RQ4. What benefits can AI and ML offer to both teachers and students in computer programming courses in higher education?

2.1. Search strategy

While it is important to consider both teachers and students in the discussion of technological tools for the classrooms, this paper intends to capture advancements and tools aiming to enhance teacher capabilities in their classrooms, i.e., focusing on the teaching process instead of the teaching and learning process. In the field of instructional design (ID), there are several models and frameworks proposed to aid the teachers in developing curriculum and courses. Among these models, ADDIE was frequently used in past literature because it offers instructors flexibility in the instructional design task (Bond & Dirkin, 2020; Stefaniak & Xu, 2020). Although the ADDIE model was proposed in the 1960s (An, 2020), its popularity in the literature reflects the benefits of the model in the education research, especially instructional design. Nevertheless, this does not mean that other instructional design models and frameworks are not suitable for this task. Rather, the instructors have to choose the most appropriate model or framework for their situation. To reflect this proposition, this study intends to be agnostic of specific instructional design models and frameworks, while maintaining the dynamic of the ADDIE model. Specifically, this study divides the teaching process of computer programming courses into four categories: course design, classroom implementation, assessment and feedback, and performance monitoring. The course design category concerns the development of curriculum, learning objectives, learning activities, and assessment rubrics. The classroom implementation category covers the execution of teaching strategy, content delivery, classroom management, and other activities related to classroom activities. The assessment and feedback category focuses on the assessment of students' ability and knowledge through exercises, assignments, labs, exams, etc. Lastly, the performance monitoring category pertains to the monitoring of students' behavior and progress throughout the course. Although the assessment and feedback category and the performance monitoring category often falls under the evaluation process in the instructional design models, e.g., the ADDIE model, this study makes distinction between these two categories by the nature of the task. While the assessment and feedback category concerns tasks that occur at discrete intervals, tasks in the performance monitoring category involve continuous actions that provide ongoing insights into students' progress and performance throughout the course duration.

After the initial identification of literature for each teaching process, the course design was removed from the keyword search due to a lack of sufficient relevant research. With recent advancements in Large Language Model (LLM), recent studies have attempted to explore its application in educational settings (Rahman & Watanobe, 2023; Dos Santos & Cury, 2023). Nevertheless, the number of literature in this specific area is limited since the applications of LLMs in education only recently emerged around two years ago after the launch of chatGPT. Discussion of relevant research on course design will be covered briefly in Section 4.2.1 for the comprehensiveness of this systematic review.

Search queries for classroom implementation, assessment and feedback, and performance monitoring were developed and used to retrieve the initial set of literature from the SCOPUS, ACM, and IEEE Xplore libraries. Search queries were designed to capture four key elements ac-

Table 1
Search queries used to retrieve literature on different teaching processes.

Teaching Process	Search Query
Classroom Implementation	(programming OR code OR coding OR “computer science” OR “computer programming”) AND (ai OR “artificial intelligence” OR ml OR “machine learning” OR supervised OR unsupervised OR “deep learning” OR “reinforcement learning” OR “data mining” OR classification OR classifier OR prediction OR predict OR learning) AND (course OR education OR class OR classroom OR university OR college) AND (gamif* OR personali* OR adaptive OR recommend* OR interactive OR tal OR “technology assisted learning”) AND (improve OR improving OR improved OR enhanc*)
Assessment and Feedback	(programming OR code OR coding OR “computer science” OR “computer programming”) AND (ai OR “artificial intelligence” OR ml OR “machine learning” OR supervised OR unsupervised OR “deep learning” OR “reinforcement learning” OR “data mining” OR classification OR classifier OR prediction OR predict OR learning) AND (course OR education OR class OR classroom OR university OR college) AND automat* AND (assess* OR evaluation OR grade OR grading OR mark* OR feedback OR hint) AND (exercise OR assignment OR “source code”)
Performance Monitoring	(programming OR code OR coding OR “computer science” OR “computer programming”) AND (ai OR “artificial intelligence” OR ml OR “machine learning” OR supervised OR unsupervised OR “deep learning” OR “reinforcement learning” OR “data mining” OR classification OR classifier OR prediction OR predict OR learning) AND (course OR education OR class OR classroom OR university OR college) AND (student AND (perform* OR “learning outcome”) AND (predict OR prediction OR predicting OR identify OR identification OR identifying OR monitor OR monitoring))

cording to the objective of this article, namely computer programming, artificial intelligence and machine learning, education, and keywords specific to each teaching process. With this design, search queries across the teaching process were similar except for their specific keywords. Table 1 provides full search queries used in this study.

2.2. Selection method

This study limited the review of scholarly articles to journal and conference articles that were written in English and published after 2010. Review papers, posters, abstracts, extended abstracts, and visionary papers were excluded from this study. Recall that this study aims to cover a consolidated perspective on the utilization of AI and ML technologies in augmenting the teaching process of computer programming courses in higher education. This raises the necessity to examine only articles that explore computational intelligent technologies to enhance and facilitate teaching computer programming at university levels in at least one of the teaching processes. Furthermore, articles that only concerned the evaluation of existing software and programs were also excluded.

Keyword-matched literature from the previous stage was manually validated for inclusion in the next review processes. Retrieved articles were first screened by their title, retaining those papers relevant to the field of AI and computer science education in higher education and excluding those that were not. Moreover, papers shorter than four pages were also excluded.

The remaining papers then underwent an additional round of screening based on their abstract content, ensuring relevance and, therefore, their inclusion in the review. While the title screening process aims to maintain recall and remove literature that is clearly unrelated to the scope of this review, there is also a possibility of false positives in the retrieved papers. Such research may discuss AI or ML and computer science education components but does not exhibit the utilization of AI and ML in enhancing the teaching process of computer science education.

After the abstract screening, only those papers that addressed the applications of AI and ML in augmenting the teaching of computer science in higher education were retained for further review. In-dept reviews and comparative analysis will be conducted on these articles that passed both title screening and abstract screening processes. Table 2 provides the full list of inclusion criteria and exclusion criteria used in this review.

3. Selection results

Fig. 1 displays the number of articles in each screening stage. The figure was produced by Haddaway et al. (2022).

The search string constructed in section 2.1 was queried on 9 April 2024. The databases returned 3,050 scholarly articles. 163 duplicates were removed from the search results. During the title and abstract screening process, 2,722 papers were excluded as they did not meet the inclusion and exclusion criteria. The remaining 165 papers underwent a full-text screening, where 46 papers were excluded. As a result, 119 papers were obtained for this systematic literature review. Note that all the articles published in 2010 - 2011 did not pass the aforementioned screening criteria; therefore, the final 119 retrieved articles were published in 2012 - 2024.

After screening the papers for eligibility, they were assessed for relevance to the corresponding teaching category. Initially, papers were assigned the teaching process from their corresponding search string, i.e., retrieved papers from the classroom implementation search string were tagged with classroom implementation. After full-text screening, papers that were found to be more relevant to other teaching processes were reclassified. Although the search string was developed to retrieve articles from classroom implementation, assessment and feedback, and performance monitoring, the retrieved articles contain 3 papers associated with the course design process. As a result, the course design process will be covered in the analysis based on these 3 papers. The number of papers in each teaching process is shown in Table 3.

Among the 119 articles retrieved, the majority of the papers came from the United States and Brazil, with 21 and 11 papers, respectively. India and China followed Brazil closely, each with ten papers. In the fifth place, Finland has six articles relevant to the research questions. The number of retrieved articles by country is shown in Fig. 2.

4. Analysis of reviewed papers

4.1. RQ1. Which areas of AI and ML can be used to augment the teaching process in computer programming courses?

The first research question of this systematic review concerns the broad area in which AI and ML methods have been proposed to support the teaching process. As discussed in Section 2, this study divides the teaching process into four distinct stages, namely course design,

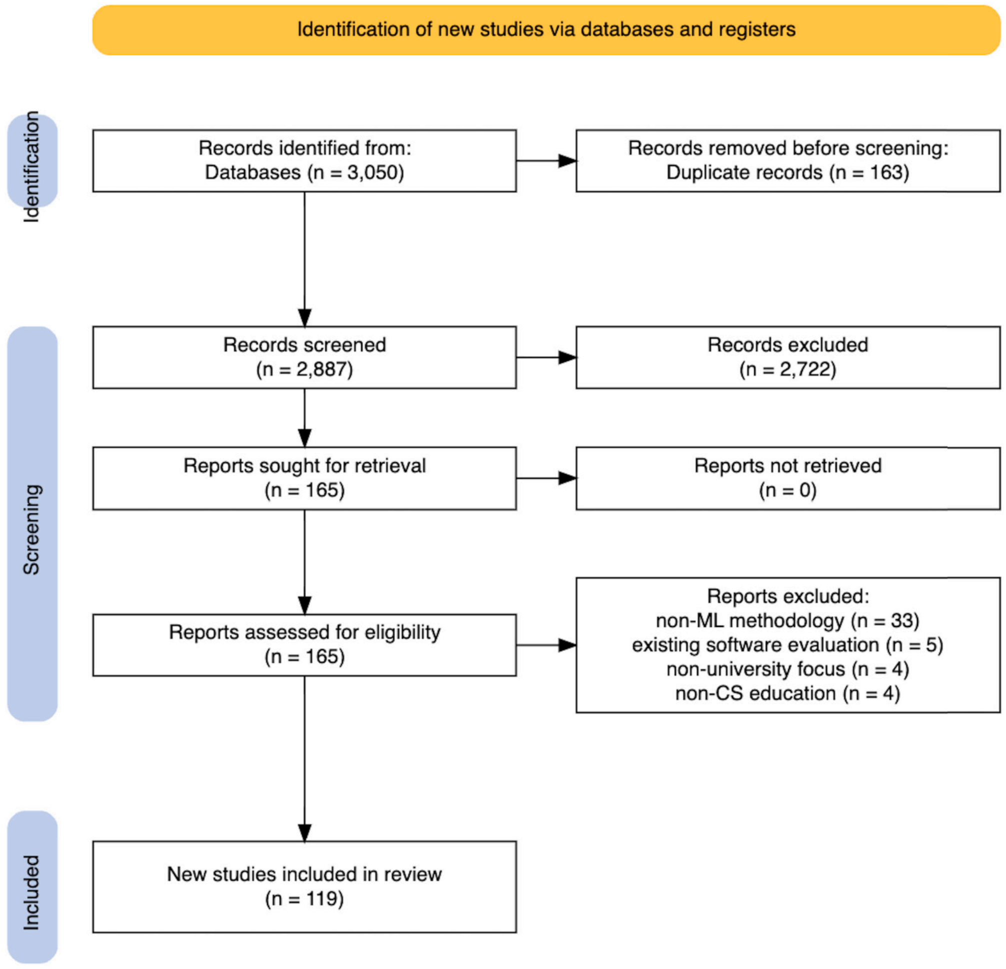
Identification of new studies via databases and registers
Identification
Records identified from:
Databases (n = 3,050)
Records removed before screening:
Duplicate records (n = 163)
Screening
Records screened
(n = 2,887)
Records excluded
(n = 2,722)
Reports sought for retrieval
(n = 165)
Reports not retrieved
(n = 0)
Reports assessed for eligibility
(n = 165)
Reports excluded:
non-ML methodology (n = 33)
existing software evaluation (n = 5)
non-university focus (n = 4)
non-CS education (n = 4)
Included
New studies included in review
(n = 119)

Fig. 1. The PRISMA flow diagram of the study.

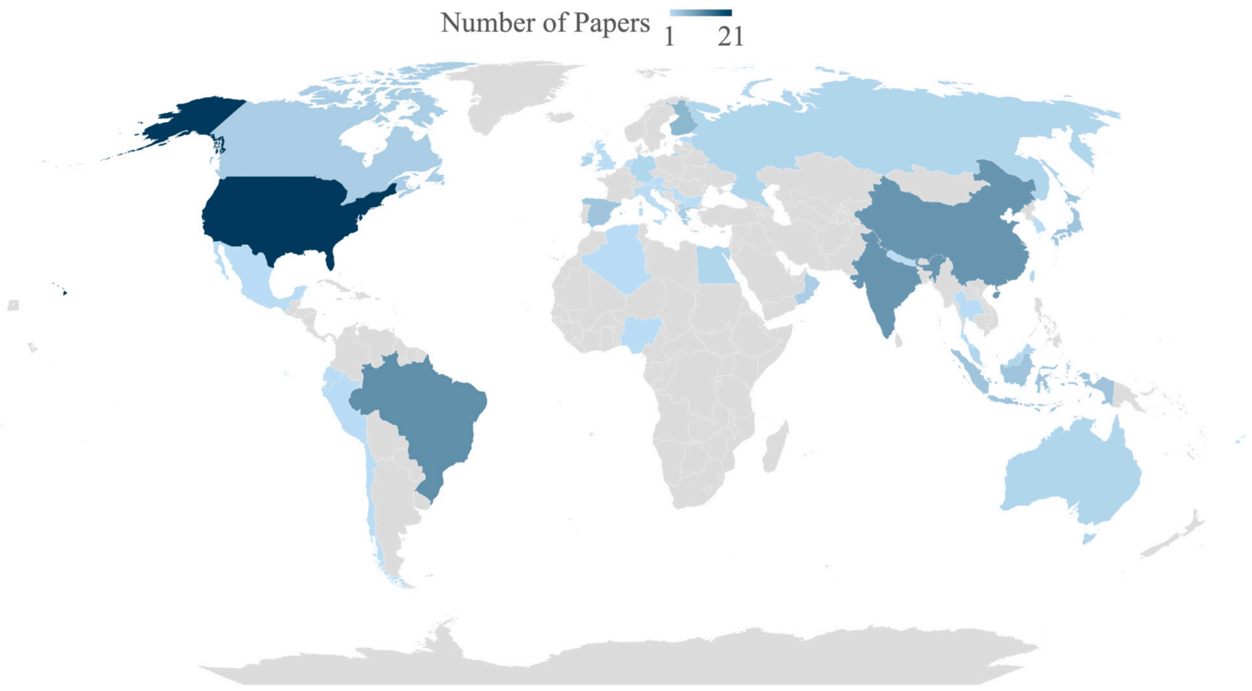

Fig. 2. The number of articles retrieved by country.

Table 2
Inclusion and exclusion criteria.

Inclusion Criteria	Exclusion Criteria
• Is written in English • Utilizes AI/ML techniques as their main methodology • Is conducted in computer programming class • Does not focus on evaluating the performance of existing software or platform	• Is a review paper or visionary paper • Is shorter than four pages • Is not in the field of computer science • Focuses on primary education and secondary education

Table 3
Number of papers in each category.

Category	Number of papers
Course Design	3
Classroom Implementation	26
Assessment and Feedback	35
Performance Monitoring	55

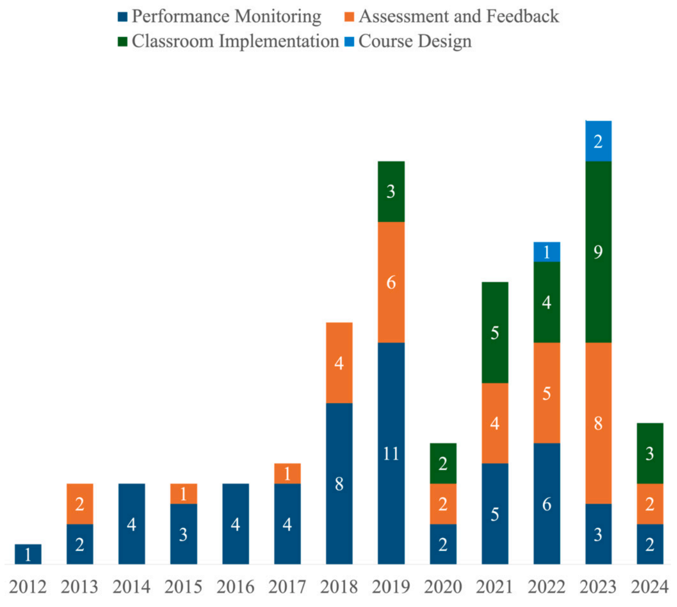

Fig. 3. Number of retrieved articles by teaching process, 2012-2024.

classroom implementation, assessment and feedback, and performance monitoring. While the initial search string was developed to retrieve articles from the latter three categories, the search result discussed in Section 3 indicated that there are three papers concerning tasks in the course design category. While it is obvious from the search result that the retrieved articles cover all categories, it is important to explore the result in detail to understand the research landscape of the topic.

The retrieved articles exhibited an interesting pattern. Fig. 3 displays the number of retrieved articles of each category between 2012 and 2024.[1] The number of literature were stagnant between 2012 and 2016. Literature in this area experienced rapid growth between 2017 and 2019, but there was a significant drop in 2020. The growth in the literature between 2017 and 2019 was from increases in both assessment and feedback category and performance monitoring category, and the new literature in the classroom implementation category. During these years, literature proposed new applications of AI and ML in computer programming education such as hint generation (Lavbič et al., 2017) and learning style classification (Yusoff & Najib Bin Fathi, 2018), reflecting the expansion of the research frontier. One of the possible hypotheses for this significant drop in 2020 is the disruption posed by the COVID-19 outbreak. Education researchers often conducts experiments to evaluate the impact of technological tool in the classroom. Many schools and educational institutions were closed during the pandemic and shifted their instructional mode to online instruction. With sudden changes in instructional approach and classroom settings, experiments planned before the pandemic cannot be conducted as these changes can affect the experiment results. This also holds true for researchers whose aim is to compare results from different cohorts. While some experiments can be conducted during the pandemic with some changes in experiment design, the results from these experiments may not be comparable to the results obtained prior to the pandemic. Moreover, this rapid transition from face-to-face instruction to remote learning led to the shift in research priority towards the implementation and adaptation of remote learning instead (Cretu & Ho, 2023). However, the numbers of literature recovered and caught up with the pre-pandemic year in 2023. It is worth to mention that research prior to the 2020 drop was concentrated on performance monitoring, but the research topics after 2020 further extend to assessment and feedback as well as classroom implementation, as can be seen in Fig. 3. The shift in the research landscape may reflect the need for research regarding contemporary issues faced in the higher education sector after the pandemic.

Although the research in the course design category began later than the other categories, they explored the new frontier of research that past literature could not achieve due to computational capabilities. In contrast to other categories, literature in course design was found in 2022, making them the newest research area in the field. With the popularization of LLMs in 2021 and the advancement in computational capabilities in deep learning algorithms, the research community has been exploring its applications and impact in their field of research, and computer programming education is no exception. In the past, designing the course structure and course content well takes a lot of effort and experience. With these technological advancements, researchers have begun to explore how to adopt these technologies in the course design process, which cannot be achieved with previous machine learning algorithms.

In recent years, research in the classroom implementation category has become the largest category in the field. Some studies have been conducted in this category in 2019 and 2020. After 2020, the number of literature in this field has grown significantly and consistently. This can be observed by a two-fold increase in 2021 compared to 2020 and another two-fold increase in 2023 compared to 2022. Researchers have explored several applications of AI and ML to facilitate teaching and learning experiences for computer programming courses. The growing number of studies in this category reflects the need to ease students' learning process.

Similar to the classroom implementation category, assessment and feedback recently became another important area of research. While research in the assessment and feedback category can be retrieved since

[1] The small number of articles in 2024 is due to our data collection being limited to the period between January 1st and April 9th, 2024.

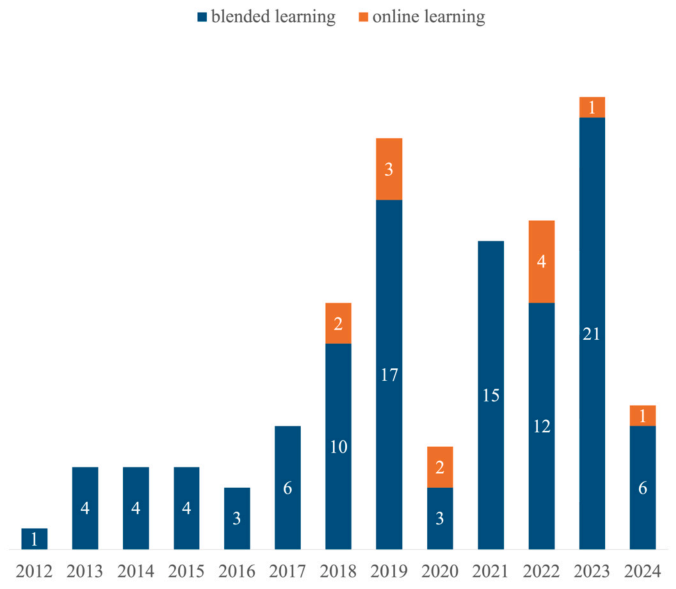

Fig. 4. Number of retrieved articles by mode of learning, 2012-2024.

2013, the number of research began to grow from 2021 in a similar manner to the classroom implementation category. Assessing students' understanding and knowledge is a crucial step to ensure that students are equipped with sufficient knowledge and working skills for the subsequent courses. Since the process of assessing a student's source code can be time-consuming due to differences in coding styles and problem-solving approaches, machine learning algorithms were used to extract insights from the program's structure. With the launch of new digital tools as well as LLMs such as ChatGPT and Gemini, students can generate code easily without fully understanding the code given by these tools. As a result, more research on how to combat the malicious use of these technologies will play an important role in the field as well as in the classroom.

While past literature in the field concentrated on the performance monitoring category, only a few recent studies were conducted in this area. Prior to 2020, researchers were interested in tasks related to monitoring student's performance. Moreover, the research in this category experienced a rapid expansion between 2017 and 2019, where the number of publications grew by twofold. However, after 2020, research in performance monitoring experienced a downward trend. This could reflect that topics in this category have been extensively explored, and the research gap has been narrowed. Nevertheless, research in the performance monitoring category has provided significant contributions to the field of computer programming education by identifying factors contributing to the prediction of student performance. Note that the complete discussion on the applications of AI and ML technology from every category is provided in Section 4.2.

To verify the hypothesis that COVID-19 is one of the disruptions posed to the research in computer programming education, the mode of learning is extracted from the retrieved articles. Initially, the authors define 3 modes of learning for comprehensiveness: traditional, blended, and online. Instruction in traditional learning is given in person, and learning activities are done in person. On the other hand, every aspect of the course in online learning is conducted over the Internet. The blended learning approach combines face-to-face learning with online learning, usually adopting in-person instruction from traditional learning and implementing digital tools like online learning. The reason why online learning is included in this analysis is that some universities may have shifted instructional mode for some of their courses to online learning during COVID-19 to adhere to the quarantine policy.

Fig. 4 provides a summary of learning modes of the retrieved articles between 2012-2024.[2] There are two observations from this figure. First, blended learning is the majority mode of learning. This is not a surprising fact, as digital tools are widely implemented in computer programming courses, from assignment submissions to online code editors. Second, the share of online learning in the retrieved articles is significantly less than blended learning. While it can be seen from the figure that some research concerned online learning before the COVID-19 pandemic, there were only two papers regarding online learning during the outbreak between 2020 and 2021. However, the number of publications concerning online learning increased to four papers in 2022. But overall, online learning still takes a very small portion of the field.

To summarize, the utilization of AI and ML spans all areas of teaching computer programming courses in varying proportions. Although the number of literature were stable between 2012 and 2016, the number of literature grew significantly between 2017 and 2019. Before 2020, publications concentrated on the performance monitoring category. After 2020, research in classroom implementation and assessment and feedback categories grew while research in performance monitoring experienced a decline. Moreover, recent research has touched upon the course design category. While the authors hypothesize that the shift in the research landscape may be the result of the rise in online learning, there is not enough supporting evidence to verify this hypothesis. To understand the research in more detail, it is important to understand not only the area they are associated with but also their applications within each teaching process.

4.2. RQ2. What applications do AI and ML take in each teaching process of computer programming courses?

As the previous research question provides a high-level perspective on the utilization of AI and ML in computer programming education, the emphasis on this research question is given to practical applications of AI and ML that have been proposed or implemented in the classroom. While it is crucial to realize that there are several ways to implement this technology within the same application depending on the instructor's goal as well as on available data sources, the analysis for this research question will only cover the applications in each category. For each application, the analysis is arranged by type of data sources used in each study to facilitate the comparison among the obtained literature. Machine learning techniques of the articles will be presented in Section 4.3. Table 4 provides a summary of the application and corresponding methodology for every retrieved article.

4.2.1. Course design

In this systematic review, course design involves the development of curriculum, learning objectives, learning activities, and other activities related to the planning stage. This category can be viewed as a combination of the analysis phase, the design phase, and the development phase of the ADDIE model. The course design process helps instructors establish what success looks like for the course and how to achieve that success with the class. Good planning benefits both instructors and learners. Instructors can use the results from this planning stage as a blueprint for their course, while learners can also use it as a reference for what to expect from the course.

4.2.1.1. Labor market alignment It is important to recognize what the industry demands for students with computer programming skills if the course intends to prepare students to work in the industry. However, grasping the understanding of the industry demand can be a complex process as these needs can change quickly due to technological change and different roles may require different skills from their applicants. To

[2] The small number of articles in 2024 is due to our data collection being limited to the period between January 1st and April 9th, 2024.

Table 4
Summary of the retrieved articles by applications.

Category	Application	Number of Papers	References
Course design	Labor market alignment	1	Chang et al. (2022)
	Lesson plan development	1	Rahman and Watanobe (2023)
	Curriculum mapping	1	Sengupta and Das (2023)
Classroom implementation	Content personalization	7	Huang et al. (2021); Rathore et al. (2021); Zhao (2022); Huang et al. (2023); Troussas et al. (2023); Wu et al. (2023); Adewale et al. (2024)
	Exercise recommendation	5	Rahman et al. (2021); Pereira et al. (2021b); Yoshimura et al. (2022); Pereira et al. (2023); Liu et al. (2024)
	Virtual assistant	4	Hobert (2019); Saini et al. (2019); Dos Santos and Cury (2023); Huang (2023)
	Question generation	4	Thomas et al. (2019); Sarsa et al. (2022a); Speth et al. (2023); Doughty et al. (2024)
	Difficult concept identification	2	Behera et al. (2020); Li et al. (2022)
	Gamification	1	Krouska et al. (2020)
	Problem categorization	1	Pereira et al. (2021c)
	Forum thread recommendation	1	Plaza et al. (2023)
	Self reflection	1	Anwar et al. (2023)
Assessment and feedback	Assignment grading	15	Stankov et al. (2013); Vujošević-Janičić et al. (2013); Barbosa et al. (2018); Rezende Souza et al. (2019); Wong et al. (2020); Luca and Chen (2020); Angelovski et al. (2021); Qin et al. (2021); Verma et al. (2021); Sarsa et al. (2022b); Ainapure et al. (2022); Acuna and Bansal (2022); Matsrostefano and Sciarrone (2022); Arhandi et al. (2023); Rico-Juan et al. (2023)
	Feedback generation	7	McBroom et al. (2018); Lobanov et al. (2019); Rubio-Manzano et al. (2019); Malik et al. (2019); Duong et al. (2022); Bahrehvar and Moshirpour (2023); Heo et al. (2023)
	Source code error detection	3	Možina et al. (2018); Gupta et al. (2019); Koutcheme et al. (2023)
	Suspicious activity detection	3	Saoban and Rimcharoen (2019); Demidova et al. (2023); Freire-Morán (2023)
	Challenging concept identification	2	Liu et al. (2023); Kerschbaumer et al. (2024)
	Hint generation	2	Lavbič et al. (2017); Roest et al. (2023)
	Code completion	1	Terada and Watanobe (2019))
	Typing pattern classification	1	Longi et al. (2015)
	Metacognition level classification	1	Beck et al. (2018)
Performance monitoring	Academic performance prediction	25	Anthony and Raney (2012); Alzoubi et al. (2013); Aziz et al. (2013); Yoo and Kim (2014); Krishna Kishore et al. (2014); Navrat and Tvarozek (2014); Carter et al. (2015); Leinonen et al. (2016); Wang et al. (2017); Kumar et al. (2016); Chiheb et al. (2017); Guerrero-Higueras et al. (2018); Ulloa-Cazarez et al. (2018); Anand et al. (2018); Tanuar et al. (2018); Sheshadri et al. (2018); Raju et al. (2019); Matetic (2019); Qu et al. (2019); Khan et al. (2019); Edwards et al. (2020); Khan et al. (2021); Pereira et al. (2021a); Sagala et al. (2022); Farghaly and El-Kafrawy (2023)
	At-risk student identification	15	Singh et al. (2014); Petkovic et al. (2014); Ochoa (2016); Costa et al. (2017); Azcona and Smeaton (2017); Richards and Hunt (2018); Erickson (2019); Shrestha and Pokharel (2019); Buschetto Macarini et al. (2019); Veerasamy et al. (2021b); Nabil et al. (2021); Arakawa et al. (2022); Au et al. (2022); Quille et al. (2023); Vives et al. (2024)
	Dropout prediction	3	Pereira et al. (2019); Naseem et al. (2019); Triayudi et al. (2021)
	Knowledge tracing	3	Jiang et al. (2020); Kantharaju et al. (2022); Liu et al. (2022)
	Student behavior classification	3	McBroom et al. (2016); Poon (2019); Chen et al. (2022)
	Learning style classification	3	Yusoff and Najib Bin Fathi (2018); Karagiannis and Satratzemi (2019); Feklistova et al. (2020)
	Student competency classification	2	Somasundaram et al. (2015); Silva et al. (2024)
	Teamwork style identification	1	Gitinabard et al. (2023)

tackle these issues, machine learning can be employed to extract skills the skills required by the employers. Chang et al. (2022) explored and proposed a skill extraction algorithm to skills required by employers. Firstly, the authors constructed a category of skills using self-reported skills from university students on a job-hunting website. Then, the algorithm takes phrases from the online job posting and classifies whether the phrases contain the coded skill or not. This algorithm leverages the natural language processing technique to analyze linguistic pattern in the job posting phrases and utilized machine learning model to classify the phrase into skill or non-skill. Lastly, a cross-sector and within-sector analyses were conducted to identify skills required for different professions. The study found a mismatch between the skills students acquired from the course and skills required by the employers (Chang et al., 2022). With better understanding of the current skills demanded by the industry, higher education institutions and instructors can integrate these understandings in their lesson plan to ensure better alignment between the curriculum and the industry needs.

4.2.1.2. Lesson plan development While developing the lesson plan is a crucial step for instructors as it lays out what content will be covered and how will it be delivered, it can be a time-consuming process if the instructor has to start from scratch with little to no assistance. This is especially true for instructors who have to teach a course they have never taught before and for inexperienced instructors who may need guidance in crafting appropriate lesson plan for their course. With the increasing popularity of LLMs, especially ChatGPT, its applicability in educational settings has been explored in Rahman and Watanobe (2023). The study evaluated ChatGPT's capabilities in computer programming education in several aspects, including but not limited to developing lesson plans, generating solutions to programming exercises, and providing explanations for the code. Specifically, the study gave several prompts in several areas to ChatGPT and evaluate its responses. The authors also explored its capabilities in assisting the learning of computer programming by asking ChatGPT to explain programming concepts, generate solution code, and check errors from source code. Furthermore, a survey on how ChatGPT can support programming learning and teaching has been conducted. The survey results showed that both students and teachers were satisfied with the supports and sugestions provided by ChatGPT. Moreover, an experiment has been conducted to evaluate ChatGPT's capabilities in real classroom settings. The result showed that ChatGPT can generate approximately 95.8% of correct code (Rahman & Watanobe, 2023). However, the authors also highlighted potential threats and risks from utilizing LLMs in the classroom that researchers and practitioners should take into consideration, such as the integrity of the assignments, reliance on generative AI tools, and development of critical thinking and problem-solving skill.

4.2.1.3. Curriculum mapping Another important element in the course design process is the alignment between the course and the program of study. In outcome-based education (OBE), curriculum mapping is an important process for instructors and course coordinators as it helps align the course objective with the program's objectives. This process is often conducted by mapping the course outcomes (CO) with the corresponding program outcomes (PO). With the absence of clear guidelines on this mapping, combined with the large volume of data that needs to be processed, the CO-PO mapping process is considered to be a complex task even for experienced faculties. As a result, machine learning technique can be applied to streamline the process of curriculum mapping. Sengupta and Das (2023) proposed an automated approach to the mapping of course outcomes and program outcomes with modern natural language processing techniques. In this study, the dataset comes from the curricula of bachelor-level Computer Science and Engineering programs from several Indian universities. The study compared the performance between traditional machine learning models with conventional features obtained from TF-IDF, Word2Vec, and stacked TF-IDF and Word2Vec with principal component analysis (PCA), and deep learning models with BERT-based embeddings. While the best traditional models with conventional features achieved the average micro F1 score of 78.23%, all deep learning models exhibited stronger performance, with Roberta giving the highest F1 score of 85.96%.

Creating the course structure and content, especially for introductory courses, is a crucial task for instructors. With careful design, instructors can create a blueprint of what students should expect to gain after completing the course. While AI and ML can assist in aligning the curriculum with the job market, instructors need to put these well-crafted plans into action to ensure the success of the course. While having a plan is a good starting point, implementing the plan is what brings it to life.

4.2.2. Classroom implementation

Although several actions and tasks fall under the definition of classroom implementation, the common goal of this teaching process is to deliver course content and nurture necessary skills to ensure that students are meeting the learning objectives set in the course design stage. Hence, tasks in the classroom implementation process emphasize execution more than other teaching processes. Because each student is unique, there is no universal classroom approach that can ensure every student achieves the desired learning outcomes.

4.2.2.1. Content personalization Among the obtained literature, content personalization has the largest share in the classroom implementation category. With technological advancement, personalized learning has become more feasible and, in turn, more popular. Tailoring instructions to student's preferences and needs can enhance their strengths while tackling their weaknesses. Since personalized learning spans across a wide range of dimensions, several applications of AI and ML within personalized learning are presented in this category, starting with personalized content. Based on the obtained result, literature leveraging AI and ML technologies in content personalization began after 2020. Some of these literature developed a personalized learning platform, which is beneficial for beginner programmers. For instance, Rathore et al. (2021) developed a personalized learning platform for programming courses. The platform consists of four sub-systems, namely material recommender, programming section, AI chatbot, and information database. A material recommendation module and AI chatbot to answer students' queries were implemented in the system to promote personalization. Student knowledge was assessed with quizzes and relevant materials will be recommended to the students based on their knowledge. After the students finish reviewing the study materials, they will move on to complete programming exercises as well as quizzes to update their profile in the system. Similarly, Huang et al. (2021) proposed an intelligent teaching system for object-oriented programming in Python course. Student progress and data are collected throughout the course, and an AI chatbot was implemented to support students' queries. Student progress is used to design personalized teaching plan for the instructor. The proposed system was implemented in the classroom and found to be effective in stimulating students' interest in learning Python programming.

Another approach to utilize student's skills in content personalization is to compare their skills with other students using collaborative filtering algorithm. Zhao (2022) developed an adaptive learning question bank system for Java programming courses. This system uses a collaborative filtering recommendation algorithm to provide personalized test exercises tailored to each student's learning progress. The goal is to enhance learning efficiency and improve student academic performance while also serving as a reference for other adaptive learning systems based on personalized recommendations. Student's rating was collected and similarity among students based on this rating was conducted. Then, recommendation was generated based on collaborative filtering algorithm. Other filtering algorithms have also been used, as seen in Huang et al. (2023). The study employed the content-based filtering algorithm to recommend video materials based on their profiles. The proposed system used self-administered sequential probability ratio test (SPRT) to collect data about students' perception of their own ability after the lecture. This self-examination and student online learning behavior are used to classify student's learning outcome. The videos are then recommended based on predicted student's learning outcomes.

Recently, other literature have explored other techniques beyond filtering algorithms. For instance, Troussas et al. (2023) constructed knowledge graph using students' prior knowledge, their learning style, and their current learning goal. This knowledge graph consists of two types of nodes, namely learners and educational entities. Associated contextual attributes of learners were assigned to give further contexts to the recommendation algorithm. To develop recommendation from the knowledge graph, learners and educational entities were represented as vectors and cosine similarity was computed. Adewale et al. (2024) explored how to incorporate student's learning styles into the content personalization. The authors proposed a design of an adaptive ubiquitous learning system by leveraging a Bayesian network. Initially, learning style of the students were extracted from the Index of Learning Style (ILS) questionnaire. Naïve Bayes Classifier was used to classify the learning styles throughout the course and update the student learning styles accordingly. Since the learning style is used to infer appropriate learning path for students with different learning styles, students who change their learning styles during the course will need a different learning path.

Some literature has incorporated theories from education into their work. Wu et al. (2023) explored the power of LLMs to generate content for programming courses based on the constructivism learning theory and cognitive load theory. The proposed system consists of five parts: programming learning program, personalized learning context, programming learning chain of thought, real-time interaction, and assessment and feedback. After implementing this model in the classroom, the proposed model was found to be effective in raising students' interest in learning computer programming. While the emphasis of the study is in the design of the intelligent system, using educational theories as foundations for the study serve as a bridge between computer science and education.

4.2.2.2. Exercise recommendation Exercise recommendation is another element of personalized learning. Since programming skill is acquired through practice, programming students should spend time with exercises to improve their understanding of programming concepts and critical thinking skill. However, it is possible that students may face with questions that are too challenging for them, which can discourage them from continue learning about programming. Thus, recommending appropriate exercises to the students can facilitate their learning process by identifying programming questions that are appropriate for student's skill level. There are several ways to tackle this application, as can be seen by the following literature. For instance, Rahman et al. (2021) developed a rule-based Online Judge (OJ) recommender system where the recommendations are based on students' knowledge which is determined by a clustering algorithm. The system first scored the submitted program and clustered these programs to understand association rules between each clusters. The resulting association rules then can be used as supporting evidence for instructors to adjust the difficulty of the programming exercises in the future.

Moving away from rule-based system, Pereira et al. (2021b) developed a behavior-based recommender system (BRS) to perform prescriptive analytics on students' information and recommend exercises. This study emphasized the importance of students' efforts, as can be extracted from students' logs and source codes into several features such as the number of attempts and the average number of lines of codes. The system will recommend problems which require similar efforts from the students. The students were satisfied with the recommendations provided by the BRS, as opposed to the random recommendations. The system exhibited the potential to support computer programming courses that use Programming Online Judge (POJ). However, the BRS system did not take students' previous knowledge into account and only used students' efforts in their recommendations. Similarly, Pereira et al. (2023) also leveraged students' efforts in their recommender system. The authors developed a hybrid human/AI recommender system where the instructor helps validate the equivalence between the target problem and the recommended problem. The system The recommended problem was evaluated based on the time student took to solve the question, the effort required by the students, and the hit rate. The hit rate is the ratio of correct solutions and the number of attempts. If the recommended problem and the target problem have similar in every dimensions, they are said to be equivalent. The experiment showed that the proposed recommender system could provide programming problems compatible with human-selected questions.

Yoshimura et al. (2022) took a different approach from the previous studies. The authors designed Waseda Online Judge Recommender (WOJR) using Waseda University's OJ database to help students that cannot solve programming exercises. The WOJR system computes similarity scores between model answers with student's submissions and returns problems with similar model answers to the students so they can practice their programming skills on similar problems. TF-IDF and JPlag were used as similarity metrics in this study because TF-IDF measures the similarity of token sequences while JPlag considers the structural similarity between source codes. The WOJR was found to increase the percentage of students who correctly answered every problem by 11.5%. Similarly, Liu et al. (2024) proposed an architecture to capture student behavior and learning patterns from their source code and use these insights to recommend programming exercises. The authors developed Programming Exercise Recommender with Learning Style (PERS) to recommend programming exercises, considering student's learning styles and preferences. Embeddings were performed on both exercise problem and student source code in order to compare the difference between the two. The resulting embeddings were used to update the latent states representing student learning style. These learning style were then used to recommend programming exercises for the students. The PERS was evaluated on four public datasets and was found to yield better performance than pedagogical methods and sequential methods (Liu et al., 2024).

4.2.2.3. Virtual assistant Since computer programming concepts and exercises can be abstract and complex, students may seek additional assistance and guidance in the early stages of learning. However, with a very high student to teacher ratio, it can be difficult for the instructor to provide sufficient support to these students. This means that it may take a long time before students can get the response from the instructor. One way to overcome this issue is to utilize virtual assistant that can provide support to the students almost instantaneously. One of the key characteristics of the virtual assistant is that virtual assistant can hold conversation with the student using natural language that the students can understand easily, and the literature regarding this application often seeks to expand this capability to optimize student's experience with the virtual assistant. It is not unexpected that early research in virtual assistant employed traditional natural language processing techniques. Hobert (2019) developed a chatbot-based learning system for introductory programming courses. The design of this system followed the design process from understanding students' and instructors' pain points to developing design principles. The chatbot can recognize students' intentions and generate answers to open-ended questions with the NLP.js library. The system was then evaluated based on its usefulness, ease of use, and practicality. The results showed satisfactory results in all dimensions. Saini et al. (2019) proposed a framework called ModBud to help teach Model-driven engineering (MDE). ModBud come with two modes, namely educators mode and student mode. With ModBud, instructors can use the bot to help create a domain model or provide advice to improve the domain model, and students can use the bot to provide feedback on their model submissions and examples of the domain model. While the high-level architecture of ModBud is similar to other chatbot with user interface, dialogue management, and database, the dialogue management of ModBud contains several components that help ModBud understand user queries and generate appropriate responses. Specifically, ModBud leveraged dependency parsing algorithms to process user queries and to form responses by extracting relevant information from its knowledge base.

With recent advancements in LLMs, literature on virtual assistant has been incorporating LLMs in their architecture. With its capabilities to generate coherent sentences, it is natural to leverage LLMs in virtual assistant application. Dos Santos and Cury (2023) studied the effectiveness of ChatGPT as a virtual peer in peer for peer instruction setting. Specifically, the study divided students into small groups, where some groups will have ChatGPT as their member. Throughout the course, students can interact with their group members such as asking for clarifications of programming concept or asking for helping debugging the code. Student performance and feedback for each group was collected via pre-test, post-test, programming assignments, and student survey. The result from this experiment indicated that students with ChatGPT as a virtual peer performed better in programming exercises and had better skills and conceptual understanding. However, the long-term effect of AI on peer instruction was not investigated and the experiment was conducted in only one course, which th results may not generalize to other courses. Besides ChatGPT, other pre-trained LLM has also been employed by researchers in developing virtual assistant. Huang (2023) leveraged BERT to design an intelligent Q&A system for several university-level courses, including computer programming, called YONA. The system consists of 4 modules, namely intelligent Q&A module, tutor recommend and match module, tutor Q&A system module, and Q&A forum module. BERT was utilized in the intelligent Q&A module to take user queries and generate corresponding response. Students can post the question to the system, and the answers will be automatically generated and provided to them. The system was deployed as a pilot project and evaluated by time taken to answer the query and user satisfaction. Overall, YONA can resolved 85% of user's problems with average response time of 30 seconds. While these two studies incorporated LLMs in a different way, the capability of LLMs in understanding and generating texts were clearly highlighted in both studies.

4.2.2.4. Question generation While the aforementioned applications in classroom implementation process centered around students, some studies have investigated how to leverage AI and ML to facilitate instructors executing their lesson plan. Crafting appropriate programming questions can be time-consuming for instructors, as they have to adapt the problems to fit the learning objectives of their course. While some inspirations can be drawn from past exercises and from other resources, it still takes time to research and look for appropriate questions. Generating questions in computer programming courses is another area that has been advanced by LLMs in recent years. Before the rise of LLMs, Thomas et al. (2019) proposed a model to generate Java programming questions based on the instructor's input. The instructor can specify the topic of the question as well as the difficulty of the question in the input. The model was designed to generate questions regarding conditionals, loops, and arrays. The model leveraged the stochastic tree-based generation algorithm in the question generation process. The proposed model can generate multiple-choice program-tracing questions at the desired difficulty with distractors, i.e., incorrect but plausible answers. The system was evaluated by students regarding the usefulness of the questions generated. Students found the system to helped with their exam as well as helped with their understanding of Java programming language.

As the power of LLMs is being realized, several studies have explored how to integrate LLMs into the question generation pipeline. Following the launch of ChatGPT-3, Sarsa et al. (2022a) explored the integration of OpenAI Codex to generate programming exercises, corresponding solutions, and explanations. The OpenAI Codex is a generative model used in generating code and code related texts. The generated exercises, solutions, and feedback were evaluated for its correctness and clarity. While the majority of the exercises created by Codex were high quality, their explanations of the code contain some inaccuracies. However, the faulty explanations did not contain major mistakes and could be fixed by the instructor. Overall, this was one of the first attempts in leveraging the power of contemporary technology in question generation. While OpenAI Codex is a specialized model for code and programming, other researchers have explored the capabilities of a more general model, namely ChatGPT, in generating programming questions. Speth et al. (2023) explored the effectiveness of ChatGPT-3 in generating Java programming exercises in classroom settings. The instructor first reviews the exercise generated by ChatGPT and can choose to accept the exercise as-is or reject the exercise and let ChatGPT generate the exercise again. The final set of exercises was given to students to solve, and feedback was collected. The majority of students felt that the exercise's structure and difficulty level were suitable, and most did not realize it was created by ChatGPT. This result reflected the capability of a general-purpose language model in performing tasks that they are not specifically fine-tuned for. With the recent release of ChatGPT-4, Doughty et al. (2024) integrated ChatGPT-4 in their automatic multiple-choice questions (MCQs) generation pipeline in Python programming course. Not only this study differed from the previous ones by experimenting with the newer model, they also experimented with different style of prompts. Specifically, the prompts used in this study contains the high-level context as well as the corresponding learning objectives of the learning module. Contexts given to the ChatGPT included the principles of designing MCQs, the definition of Blooms' Taxonomy, course description, desired question type and examples, and the desired format of the output. This was done to ensure that the generated questions are consistent with the learning objectives the instructors aim to achieve with these questions. AI-generated questions were found to be of similar quality to questions from the instructors, and those questions closely aligned with the learning objectives more than those generated by humans.

4.2.2.5. Other applications Beyond the above categories, several studies have explored the capabilities of machine learning in other applications under classroom implementation. While there are few studies focusing on these topics, the application proposed by these studies can have significant impact in the classroom and student's learning journey. For instance, Krouska et al. (2020) integrated gamification with tutoring by developing a mobile game-based learning (MGBL) application for computer programming. Players will compete with each other on adaptive quiz testing their knowledge in computer programming. The system will recommend players with comparable competitors based on their learning modality, previous knowledge, current knowledge, and their misconceptions from previous quizzes. The system was found to be satisfactory from both student and instructor perspectives, highlighting the effectiveness of the competitor recommendation engine.

One benefit of face-to-face instruction is that it is possible for instructors to observe student nonverbal behaviors to identify which students may struggle with the class. Behera et al. (2020) further explored this opportunity by developing an automatic non-verbal behavior detection system from video. The study recorded student's facial expression and hand-over-face (HoF) gestures throughout the lecture and exerciser period. Then, HoF gestures, head movement, and other facial features were extracted from these video. The result showed that there was a significant increase of HoF gestures when the difficulty of the exercise increase. The result of this study highlights the importance of understanding student's body language to ensure student engagement and intervene when students need further assistance.

Doing exercises is an integral part of learning, especially in computer programming where students are expected to complete several programming exercises to develop familiarity with the language as well as develop their problem solving skills. However, beginners may can get overwhelmed with these exercises as some problems may be too challenging for their skill level. Pereira et al. (2021c) proposed an algorithm to predict topics of programming online judge (POJ) using random forest model with Word2Vec and contextual paraphrase with F1 score of approximately 86%. The algorithm takes the string of problem instruction and outputs its corresponding category. This algorithm can be used to assist self-directed programming learners to navigate through their learning journey by selecting problems that are appropriate for their learning goals.

Moving beyond lecture and exercises, discussion forum is a great place to understand how students learn. Li et al. (2022) investigated the online discussion forum from the Chinese University MOOC platform to understand how students interact with each other. The algorithm can extract students use the platform such as provide definitions, describe facts, or describe the relationship between methods. Moreover, another algorithm was implemented to extract student's sentiment in the discussion forum. The results from these algorithms serve as supporting evidence for instructors and other relevant stakeholders to adjust the learning module and exercises.

Similarly, Plaza et al. (2023) also investigated the discussion forum data. However, this study focused on developing a forum thread recommender so students can see who have asked similar questions in the forum and can contribute or ask for further clarifications. The author developed a content-based recommender based on student's queries. The study experimented with several feature representation techniques, namely bag of words, bag of key phrases, and embeddings. Bag of key phrases was found to have the best performance, with precision at five of 80% when using relaxed criterion.

While self-reflection can be a great tool for students to assess themselves, writing good reflection takes a lot of practice. Good reflection should be objective and specific in what areas students to improve. To facilitate the self reflection process, Anwar et al. (2023) developed an application to calculate the specificity of the student's reflection. The standard version will only provide the level of specificity, while the adaptive version also provides meaningful feedback on how to improve the self reflection, i.e., the adaptive version method uses scaffolding technique to help students write better reflection. The experiment was conducted to compare the specificity of the reflection written between these two versions of the application. It was found that the students using the adaptive version of the application receive higher specificity score in their reflection than the students using the standard version of the application. This result reflected the role of using scaffolding in assisting students successfully performing speciic tasks.

Translating learning objectives and lesson plans into actionable plans and executing those plans in the classroom is how instructors deliver course content and knowledge to students. Although AI has proved beneficial in completing these tasks, it is also crucial to assess students' understanding and skills based on the delivered content. Without appropriate assessment tools, instructors cannot measure students' knowledge and may miss opportunities to revise their strategy for the remaining portion of the course.

4.2.3. Assessment and feedback

Although assessment and feedback play a pivotal role in the computer programming course, it also takes a lot of the instructor's time and effort to perform these tasks effectively and meaningfully. Furthermore, these learning evaluation processes can provide useful insights for the instructors on students' understanding of technical concepts and their room for improvement. With an increasing number of students enrolled in computer programming courses, performing these tasks effectively is becoming more difficult for instructors. This raises the need for new tools and methods that can assist instructors in assessing students as well as providing meaningful feedback.

4.2.3.1. Assignment grading Grading computer programming assignments can be done by comparing the program's output of the submitted program with the expected output or investigating the source code to check the program's syntax, code quality, as well as compilability. The former approach is called dynamic assessment, which is usually done by creating several unit tests; the latter is called static assessment.

Since the automated assessment system can produce false positive results and false negative results, several studies proposed a semiautomatic assessment system where some processes are automated and some processes are still handled by the instructor. One approach to semiautomatic assessment is to automatically select programs that need instructor's reassessment by clustering every submissions for further analyses. Stankov et al. (2013) developed a semiautomatic assessment tool called EMAx which can cluster source code that received a weak score from an automated grading system. The submissions were first parsed into trees, then hand-crafted features were extracted from these parse trees. The clustering algorithm takes vectors of program features and provide clusters as output. They found that clustering programs based on their characteristics can be used to assess whether the program likely embodies a good implementation or an approximately effective algorithm for the given problem. Programs that did not pass all the test cases but were put in the same cluster as the programs that pass all the test cases will be reassessed by the instructor. Angelovski et al. (2021) addressed the inability to handle uncompilable code of EMAx and developed an improved application named DEMAx. In contrast to the EMAx where the program takes parsed trees of the programs as input, the DEMAx takes source code as input and conducts static analysis on the source code. The application still represented the programs as attribute vectors and utilized clusters of source code to distinguish submissions that may require an instructor's manual inspection, e.g., submissions that are uncompilable but the program's structure is similar to the correct solution. In a similar manner, Barbosa et al. (2018) also clustered students' source code to group similar programs together for instructors to evaluate. The study conducted an experiment to compare the evaluation result of the clustering approach with the evaluation from the specialists. The proposed method was found to provide a higher agreement than that obtained between the experts, with Cohen's Kappa of 0.76.

Vujošević-Janičić et al. (2013) expanded the data sources for assessment by combining automated testing results, software verification, and program similarity to predict the submission's score. Since the use of dynamic assessment alone cannot ensure functional correctness of a program or the absence of bugs, software verification and program similarity were employed to solve this issue. Final assessment score was a linear combination between the results from these three measures. Multiple linear regression was used to estimate the weight corresponding to each measure. Verma et al. (2021) extended the similarity measurement by comparing the computed similarity with the perceived similarity by the instructor. In doing so, the system can map the predicted score to the scores that are computed using the rubric. The proposed method used Support Vector Machines (SVM) to learn appropriate scaling factors for different levels of similarity by the system. The mean absolute error of the proposed system were below 0.06 for all seven types of programs that were tested. Arhandi et al. (2023) proposed a system that first computes the similarity score between the submission and every reference solution. If the answer passed the threshold, it will be run against the reference solution with the highest similarity to see if the result is the same.

Ainapure et al. (2022) integrated both dynamic assessment and static assessment in the grading model by considering the results of unit testings, time complexity, space complexity, program errors, and source code features such as coding styles and readability. These features were extracted from the source code and scoring was calculated based on existing rubric. A random forest classifier was then trained on these data to predict scores for the other submissions. By incorporating other features beyond the results of the test cases, the predicted grade can serve as a more meaningful metric for the students. Sarsa et al. (2022b) proposed a sequence to automatically grade student submission where machine learning is used to predict the existence of the problem in the source code. If the submission is the new version that has never been tested before, static analysis will be conducted to ensure the program can execute. Then, machine learning model will use the features generated from the static analysis to predict whether the submission has any problem. If the model can detect no problem, the submission will be ran against test cases to calculate the score.

Instead of parsing source code into abstract syntax trees, many studies have investigated other representation technique in assignment grading, mainly embedding technique. The grading model proposed by Rezende Souza et al. (2019) takes source code as input and creates corresponding embedding for the program by using a skip-gram. Then, a convolutional neural network (CNN) is employed to classify the embeddings into letter grades from A to F. The proposed system can output possible letter grades from A to F with 74.9% accuracy. Similarly, Matsrostefano and Sciarrone (2022) also employed embedding techniques on students' source code before ranking them based on similarity with the teacher's code. Instead of using skip-gram, the study employed doc2vec to embed the source code. Then, dimensionality reduction was performed through principal component analysis (PCA) in order to similarity with the instructor solution using Euclidean distance. The programs are then ranked based on the distance and the programs with better ranking will receive better score. The method proposed by Qin et al. (2021) used the encoder-decoder model with attention layer to perform assignment grading. The system both semantic features extracted from the source code and the result from the black-box testing to provide grades for students. The semantic features were extracted using Neural Code Comprehension (NCC) model. The result from the encoder-decoder model will be concatenated with the result from the black-box testing, then it will be passed through a softmax function to calculate the resulting grade.

While summative assessment is dominant in assignment grading, formative assessment is also helpful in assessing student's ability. Acuna and Bansal (2022) leveraged student activity trace from an automated assessment system to conduct formative assessment on the student source code. The study extracted the number of test cases passed for each submission to construct a Student Action Graph (SAG), which is similar to a Markov chain but without the independence assumption, and to calculate fragility score (FS), which can identify the parts of the assignment that the students find challenging. Students submissions were clustered, then the SAG for each cluster was constructed and the FS for each cluster was calculated. This attempt in formative assessment opens the door for in-depth assessment that summative assessment cannot capture. Students' behavior extracted from the Online Judge (OJ) system can also be used to assess their submission. Rico-Juan et al. (2023) integrated multi-instance learning, traditional machine learning, and explainable AI to predict whether the student submission pass or fail the Online Judge (OJ) system based on their behavioral pattern extracted from OJ activities. The proposed method utilizes bags of elements where the bag is labeled positive if at least one element in the bag is positive, and is labeled negative otherwise. Post-hoc explanations are implemented to provide explanation for the predicted result.

Besides the aforementioned methods, other techniques have also been discussed in the literature regarding assignment grading. Wong et al. (2020) developed a machine learning algorithm to extract matching rules from hierarchical program output structure (HiPOS). The proposed algorithm can generate matching rules for both token level and block level. The resulting matching rules can be used to identify different types of acceptable program output. Explainable AI was used by Luca and Chen (2020) to develop the Visual IoT/Robotics Programming Language Environment (VIPLE). The proposed framework integrates a modified version of π-calculus and Hoare logic to perform program verification and rules from the solution files. The system can perform automated assessment of student code and provide grade for them.

4.2.3.2. Feedback generation Another application of machine learning in the assessment and feedback process is feedback generation. While the grade received can reflect student's ability, grade alone is not sufficient to let students understand what they did well and what they can improve. This issue is present in programming course where programming tutor is employed with automated test cases. Students may find ways to solve their program just to pass the pre-defined test cases without fixing the program to work as intended. McBroom et al. (2018) proposed a method to cluster student submissions based on their functionality, assisting the instructor to identify common mistakes in student submissions. Interaction graph and sequential pattern mining were also employed to understand how students respond to the test case results given by the tutoring system.

Another issue related to providing feedback is that manually providing feedback can be time consuming. As such, automated generation of feedback has been explored in the obtained literature. Extracting the structural similarity of the source code can be used to facilitate the feedback generation process. For instance, Lobanov et al. (2019) clustered incorrect solutions to identify frequent errors made by students and used these errors to identify errors in new submissions. The only step that requires expert intervention is the labeling step after the clustering step. By utilizing both unsupervised learning in clustering frequent errors and supervised learning in classifying error types for new submission, this reduces the time needed to analyze the source code from the first step. Understanding error type alone may not be enough to guide students in the right direction, formative feedback that sufficiently provide suggestions for the students are more appropriate. Duong et al. (2022) developed a web-based tool called AP-Coach, which can compute structural similarities between students' submissions and reference solutions. AP-Coach is able to generate formative feedback by comparing student submissions with multiple reference solutions and generate feedback and pseudocode from the most similar reference solution. This can encourage. The Pseudogen was found to perform better than Denigma in automatically generating pseudocode, with 71.6% accuracy. Similarly, Bahrehvar and Moshirpour (2023) extracted the data flow from a source code by parsing the source code to abstract syntax tree (AST) and identify variable sequence as well as variable relations in the AST. The similarity metric was calculated by GraphCodeBert, a pre-trained model for programming languages. Feedback will be provided to students if the similarity score falls below the acceptance threshold.

Although feedback helps students understand how they can improve on specific exercise, it cannot provide a comprehensive profile of the students. Rubio-Manzano et al. (2019) proposed a method to extract linguistic descriptions and execution traces to generate personalized behavior profiles. This approach can provide tailored feedback on algorithm implementation, which help not only students to understand their mistakes but also instructors to gain a deeper understanding of their students. Malik et al. (2019) incorporated an approach called neural approximate parsing with generative grading (GG-NAP) in their feedback system to predict student decisions based on a given solution, enabling detailed automated feedback. The proposed method first models the student cognition by using student decision process (SDP) and Idea2Text, and then uses this cognitive model to infer feedback from student submissions. The generated feedback can highlight the part from the student submission that is the root cause to their misunderstanding.

Previously, feedback was generated based on other reference submissions. Recently, literature has explored an approach to generate feedback without the reference program. Heo et al. (2023) developed a transformer-based feedback generation system called REFERENT, leveraging transfer learning to generate feedback without a reference program. The proposed method for feedback generation will first localize the bugs in the programs then generate appropriate feedback. The author first developed a pre-trained model for feedback generation from open-source repositories. Assignment submission data was then fed to this pre-trained model to train the model to generate the feedback. The model takes string of student submission as inputs without considering the program structure and other abstract information. The experiment showed that REFERENT can generate feedback for more than 30% for incorrect programs without reference.

4.2.3.3. Source code error detection As novice programmers tend to make mistakes in their programs, enhancing the ability to detect errors in their source code can help streamline the assessment process. While the instructor may be able to manually catch these errors, it can be a time-consuming process when the class is large. Možina et al. (2018) proposed a semiautomatic assessment system to classify frequent students' approaches and errors in their submission and to facilitate the interaction between the instructor and the system to further enhance the classification. The system takes student source code as input and transforms it to abstract syntax tree, where the patterns in the trees are used as model attributes. The study employed the argument-based machine learning (ABML) method, where expert and the model collaboratively adjust the classification until the performance of the model is good enough. The final model achieved better accuracy that the models learned from human-defined features or from automatically extracted features. However, it is worth to mention that the ABML method can take a long time to reach the desirable performance because the model has to adjust based on the expert argument. Gupta et al. (2019) developed NeuralBugLocator to identify bugs related to failing tests without the need to execute the program. Similar to the previous study, the source code was first represented as abstract syntax trees and the trees are passed through the convolutional neural network (CNN) to classify whether the source code will pass the test cases or not without running the program. If the program is classified as buggy, bug localization will be performed to identify where the students should fix their code by comparing buggy program with the correct program.

Koutcheme et al. (2023) extended the previous works by incorporating program repair algorithms in their method. The authors explored methods for automatic program repair to support debugging using edit distances and the ROUGE metric. When the students submit buggy solutions to the system, the system will run it through automated repair tools. The resulting repair options are then ranked based on how far each repair option is from the original code using a distance measure. Several distance measures were experimented in this study, e.g., string edit distance, sequence dit distance, and tree edit distance. The study also investigated the performance of using NLP metrics in ranking the repair candidates instead of traditional edit distance. The result from the experiment showed that the ROUGE score, originally developed for evaluating the performance of machine translation task, had desirable performance in evaluating and selecting repair candidate. This study highlights the strength of NLP metrics which is the ease of interpretation.

4.2.3.4. Suspicious activity detection Programming skill, just like any other skills, can only be improved through practice. While the instructor can encourage their students to consistently practice by providing several programming exercises, they cannot completely prevent the students from taking shortcuts, e.g., look at the solution on the internet or copy the code from their peers. Recent studies were conducted to automatically detect suspicious activities from students. Saoban and Rimcharoen (2019) proposed a model to identify the original copy of the source codes from the plagiarized programs. The difficulty of the plagiarism detection lies in the fact that it can be unclear which source codes is the original copy and which copies are the plagiarized version. The study synthetically created several plagiarized code from the solution to the programming exercises. Then, similarity score between two programs and other related metrics were calculated such as the number of similar lines and the number of similar tokens. The best performing model was decision tree, which can identify the original copy with 82.6% accuracy. Besides similarity score, Freire-Morán (2023) also investigated the abstract syntax trees (AST) of the source code in detecting plagiarism. The authors integrated machine learning algorithms into AC2, an open-source plagiarism detection, to detect similarities between programs from the online judge system. The source codes were represented using ASTs and the trees are used to calculate the similarity metrics.

Besides plagiarism detection, it is also helpful for instructors to identify other suspicious actions from their students by looking at how they interact with the learning management system. A model developed by Demidova et al. (2023) leveraged the messages generated from verifying students' submissions in the Digital Teaching Assistant (DTA) to classify students into different types based on their behavior as well as detect suspicious students, e.g., those who only interact with the DTA system at the end of the semester. The hierarchical clustering was performed to identify high-level approaches the students take to complete the programming exercises. The authors also analyzed the timing of the student's submission. If the students did not interact with the DTA system at all and then submitted every exercise on the last week of the semester, that student is considered suspicious. Extracted student activity features were also clustered using DBSCAN to identify patterns of suspicious students and typical motivated students. This study highlights the utilization of activity log in computational education research which contains granular details that can be extracted into meaningful insights for the instructor.

4.2.3.5. Challenging concept identification As beginner students learn programming, they often make mistakes in their code because they misunderstand some concepts, e.g., memory management and pointers. While it is possible for the instructor to gauge which concepts are challenging for the students, it can take time and experience for the instructor to manually identify those concepts. Detecting errors in the source code also allows instructors to identify common challenges faced by students in learning computer programming. For instance, Liu et al. (2023) extracted frequent programming errors from students' submissions and compared these errors across different groups of students based on their performance levels. Student source code was represented using abstract syntax trees (AST) and were clustered by DBScan algorithm. The linter messages for each cluster were extracted and visualized through word cloud. The resulting word cloud showed that low-performing students and efficient students face different issues in their code.

Similarly, Kerschbaumer et al. (2024) determined the importance of the concepts taught in the C programming course based on errors in student's source code. The proposed framework ran five code analysis tools on students source code to extract several code metrics. Then, these metrics were used as features of the random forest classifier to classify the challenging concepts found in the source code. Since the aim of both studies in to automatically identify challenging concepts faced by students, feature extraction process was done through existing code analysis tool. Future research could potentially investigate alternative approaches to extract specific features from the source code that cannot be produced by these tools.

4.2.3.6. Hint generation While programming exercises are essential in the learning journey, beginners may get stuck on these exercises because they do not know how to approach the problem. Providing hints for programming exercises can guide students to the correct answers without directly giving the answer to them, encouraging them to develop problem solving skill. Traditional technique was employed by Lavbič et al. (2017), where a recommender system to automatically generate hints using past exercise solutions. The system's goal is to ensure that the students are moving in the right direction. The proposed system employed a Markov Decision Process (MDP) to model student's learning process. Specifically, the MDP encoded student's knowledge from the submitted code and the traversal on MDP is used to identify the most appropriate hint.

Naturally, recent literature has utilized a more modern method in generating hints. Roest et al. (2023) explored the utilization of a large language model (LLM) to automatically generate a hint from a student's source code. The study employed OpenAI's gpt-3.5-turbo model for this experiment. The LLM was integrated with a web interface where students practice their programming skills through various exercises. The generated hints were evaluated by both the students and the experts on ten criteria, spanning from correctness and usefulness of the hint to the tone of the response. While LLM-generated hints still contain some level of misinformation, the hint is personalized for each student and is also specific on what the students should do next. While the power of LLMs in generating hint was exhibited in this study, there are several areas that future research can investigate further such as reducing misinformation from LLMs and inferring student knowledge from the hints in order to develop student model.

4.2.3.7. Other applications It is clear that extracting appropriate features from student source code can streamline the process of assessing students' abilities. However, the obtained literature contains some studies that took a different angle in analyzing these data. For instance, Longi et al. (2015) attempted to identify students based on their typing patterns. The study defined three levels of typing patterns, where students can have one or more levels of typing patterns. These levels were used to measure similarity between students using Euclidean distance, and the nearest neighbor classifier was used to classify students in a test set. Although the authors noted that the experiment results did not reach the accuracy levels of other classifiers, this study explored the data that not many studies had utilized before.

Besides assigning grade to the source code, extracting higher level representation of student knowledge from source code has been explored. Beck et al. (2018) proposed a method to extract metacognition level of the students from their source code by looking at how the students structure their code. Specifically, the authors looked at how students use source code comment to communicate their thought process throughout the program. Meaningful comments can help students understand what the code should behave, which is very useful when they come back to review their code in the future. The multinomial logistic regression was employed to classify student source code into three categories, namely unitization, every-line, and insufficient.

It can be difficult for beginner programmers to create the correct program from scratch. Terada and Watanobe (2019) explored a way to help alleviate this issue by developing a machine learning model that can perform code completion given the source code. The proposed method utilized Long Short-Term Memory (LSTM) network to predict within-vocabulary words and pointer network to predict identifiers in the source code. The method first normalizes the source code, constructs relevant vocabulary using pre-defined word groups, then converts the source code into sequence. Neural language model was used to predict the next words given the sequence obtained from the preprocessing step, then the switch classifier was used to determine whether the predicted word is within-vocabulary word or an identifier. This approach could be implement complementarily with the automatic grader to transform the source code into compilable code before garding.

It is important to note that integrating machine learning algorithms in the extraction of features from students' source code helps instructors assess students' understanding and provide personalized feedback better and quickly. However, the success of students in computer programming courses does not depend solely on one specific assignment. Since students' performance can differ across assignments for several reasons, instructors should keep track of students' trajectories to support their learning to ensure their success. The analysis of the obtained literature regarding this issue is provided in the next category.

4.2.4. *Performance monitoring*

Monitoring students' performance poses challenges and complexities to instructors that are different from the assessment and feedback process. While student's academic performance and attributes such as exam scores, grades, and attendance are collected during their study, these data points reflect their past performance but not their future trajectories. Without predictive analytics, instructors can only provide defensive measures for students, e.g., allowing students to retake failed quizzes and exams, but not preventive measures that can help students overcome their challenges before reaching major academic milestones. While it is obvious that these tasks provide great benefits when they are performed at the early stage of the course, the main challenge lies in the lack of available data points to make accurate predictions. This translates to the use of machine learning algorithms to extract information from different sources to ensure predictive power on student performance.

4.2.4.1. Academic performance prediction Predicting student performance has been studied and explored extensively by researchers because it helps instructors grasp the overview of their class. Course grade and GPA were commonly used metrics in this application due to availability of the data. One of the commonly used data source in predicting academic performance is students' prior academic performance, specifically grades from other courses. Anthony and Raney (2012) developed a model to predict student's performance given grades in all other courses by leveraging the fact that computer science curricula usually consist of a complex prerequisite structure in the form of a direct acyclic graph (DAG). The authors constructed a Bayesian network consisting of courses from the computer science curriculum where edges representing prerequisite structure between the courses. The proposed network was able to predict the students' grades given grade in all other courses.

Since Bayesian network requires independence assumption, other studies have investigated how to use other methods in predicting students' performance instead. Krishna Kishore et al. (2014) incorporated students' previous academic records, attendance, working nature, discipline, and degree of intelligence in predicting students' grade in the current semester. Instead of constructing a Bayesian net, the authors discretized the data and used these transformed data as features in the classifiers. After validating the performance of several classifiers, multilayer perceptron (MLP) was found to have the best performance with 97.37% accuracy, outperforming Naïve Bayes and decision tree. Similarly, Chiheb et al. (2017) also discretized the grade data to be used as targets for the classifier. Besides using grade as a target variable, the authors also included two additional target variables, namely success and diploma. Both targets are binary variables, where success indicating whether the students will pass or fail the specific semester, while diploma indicating whether a student will graduate or not. Instead of the MLP, the study employed decision tree model due to its explainability. The proposed model achieved an average accuracy of 79.55% for predicting students' grade.

Early prediction of students' outcomes using students' past academic performance enables instructors to provide targeted interventions to those whose predicted result indicate they may underperform in the course. Anand et al. (2018) proposed a recursive clustering technique to track students performance throughout the semester and to provide assistance for students who need further assistance in their study. The authors first implemented k-means clustering to group students into three clusters using their past academic performance and current coursework results. Clusters with low performance were given tailored learning resources. The students were re-clustered after two weeks of tailored support to track students' progress. This process was repeated three times to ensure that low-performing students have moved up to a better performing cluster. After the intervention, the authors predict the outcomes of the test result using linear regression. This early prediction of students' performance using their performance from previous courses were helpful in providing additional supports.

Since the first year content serves as foundation knowledge that student needs to comprehend intermediate and advanced courses in the remaining years, it means that these courses are crucial in determining their final-year outcome. Based on this rationale, Tanuar et al. (2018) proposed to use students' performance in the first semester to predict final-year GPA. The authors collected the grades from algorithm, English, character building, programming language concepts, calculus, and discrete mathematics courses taken in the very first semester. The authors experimented predicting students' final-year GPA with generalized linear model (GLM), deep learning model, and decision tree. The experiment showed that the deep learning model achieved the highest accuracy among the three models with 67.6% accuracy.

While previous work heavily relied on previous academic performance, Khan et al. (2021) expanded the set of attributes used in predicting students' performance to students' basic information and current course load. Specifically, the authors included students' gender, nationality, sponsorship status, declared major, as well as number of hours registered in the current semester in the prediction model. The study evaluated the classification performance between Naïves Bayes, Support Vector Machines (SVM), k-nearest neighbors (kNN), and decision tree. The experiment showed that the k-NN classifier had the best performance with 80.9% accuracy. In a similar manner, Aziz et al. (2013) also incorporated students' background information such as gender, race, parental income level, and hometown in the prediction process. The authors employed Naïve Bayes classifier to predict students' first-semester GPA. Considering that there is no academic records for the first semester, students' demographic is the only available data source so it was used as proxies for students' academic performance. Khan et al. (2019) is another study that utilized students' past academic performance in predicting students' final outcome. After feature selection process, students' gender, CGPA, and score from the first test (accounted for 15% of the total grade) were selected to the classification algorithm. The experiment indicated that the decision tree achieved the best performance with 88% accuracy in predicting students' final outcome.

While students' past performance have been heavily utilized in the literature, it only reflected the outcomes from the past. Predicting students' academic performance should incorporate more data related to students' behavior and effort in learning the materials. As such, Kumar et al. (2016) collected several behavioral attributes of the students such as time spent on social networks, library usage, and problem solving efficiency. The authors employed the M5P, a binary regression tree model, to predict students' performance based on these behavioral attributes. The proposed model was evaluated against Adaboost, Naïve Bayes, RBF, and J48 decision tree. The experiment showed that the M5P model achieved the best performance with 97.17% accuracy. Using behavioral attributes in the prediction process enabled instructors to provide actionable guidance to the students to adjust their specific behavior for better performance. Similarly, Raju et al. (2019) not only collected students' demographic information but also their number of study hours per day and internet usage per day in the proposed method. The Naïve Bayes classifier was built to predict students' performance. The proposed method was able to predict students' performance with 84% accuracy.

While behavioral attributes are essential in predicting students' performance, manual data collection process such as survey and questionnaire can be expensive and the data may be biased. As a result, literature have explored alternative approaches to extracting students' behavior without relying on survey and questionnaire. For instance, Alzoubi et al. (2013) extracted student's activity log from iList, an intelligent tutoring system that is used to facilitate the learning of linked lists. The extracted activities were aggregated by counting the number of each actions performed by each student. The extracted data was then used to predict students' grade with logistic regression. The proposed method can successfully predict students' grade with 87% accuracy. Similarly, Carter et al. (2015) also collected and extracted students' actions from the IDE. The study captured and modeled the students' behavior with the normalized programming state model (NPSM). The constructed NPSM accounted for 36% of the students' final course grade. Moreover, the proposed method was found to be better at predicting student performance than the error quotient and Watwin score.

As many higher education institutions have been implementing learning management system (LMS) to facilitate the teaching and learning experience, it is possible to extract students' behavioral attributes and patterns from the data generated from the LMS. Sheshadri et al. (2018) extracted the interaction log from Moodle and WebAssign for discrete mathematics course and Java programming course to analyze students' behavior and predict their final grade. It was found that the number of sessions generally increased near the assignment due dates and the exam dates, indicating that students were deadline-driven. Moreover, failing students generally had lower actions than those who pass the class. Logistic regression, decision tree, and k-nearest neighbors were evaluated in the performance prediction task. k-nearest neighbors classifier was found to have the best performance in predicting final grades for Java programming with F1 score of 77.78% course while decision tree was the better model for discrete mathematics course with F1 score of 80.77%. Qu et al. (2019) explored the use of MOOC log data to predict student performance and mastery of knowledge points. The authors extracted features related to assignment submission such as submission time, completion duration, and number of submissions that pass all test cases in the first try. The study formulated the problem to be a multi-task learning problem and employed a multi-layer long short-term memory (LSTM) model with mechanism for these problems. The proposed method was able to predict students' performance with 92.52% accuracy.

Besides LMS, students' interaction with the version control system was also utilized in the literature to predict students' performance. Guerrero-Higueras et al. (2018) extracted features from students' interaction with the GitHub classroom and used them to predict students' performance. The extracted features included the total number of commits, the total number of days with commit operations, the average number of commit operations, the number of lines of code added, and the number of lines of code deleted. The authors evaluated the prediction performance among adaptive boosting, classification and regression tree (CART), k-nearest neighbors (kNN), linear discriminant analysis (LDA), logistic regression, multilayer perceptron (MLP), Naïve Bayes, and random forest. The random forest classifier was found to achieve the best accuracy of 0.8, outperforming other classifiers.

To fully leverage the behavioral attributes, it is important that the prediction results are explainable. To this tend, Pereira et al. (2021a) leveraged fine-grained interaction data from an online judge system to predict and interpret student performance. The authors analyzed several behaviors such as procrastination, time spent on programming, error rates, and submission patterns. The study employed XGBoost algorithm to predict whether the students will pass the class or not. The model was able to predict students' performance with average accuracy of 81.3%. Moreover, the study used the SHAP framework to investigate the importance of each features in the prediction result.

Submitting assignments and completing exams are not the only ways students interact with the course. Online forum is where students ask questions and share their ideas with the classmates. Literature has explored the use of these interactions in performance prediction task. For instance, Yoo and Kim (2014) attempted to predict group project performance by investigating students' roles in online discussion. Specifically, the authors analyzed linguistic features and participation patterns with Linguistic Inquiry and Word Count (LIWC) and Coh-Metrix. Support Vector Machines (SVM), decision tree, and Naïve Bayes were evaluated in classifying students into information-seeking and information-providing. SVM was found to be the best performing model for role classifier. Moreover, those with information-providing behavior was found to perform better than those with information-seeking behavior. Similarly, Ulloa-Cazarez et al. (2018) proposed to incorporate students' forum activity such as the number of times the students view the forum, post messages in the forum, and created the forum threads. The study evaluated the prediction performance among genetic programming, linear regression, and polynomial regression. The result showed that genetic programming was more accurate in predicting students' performance using interaction log than the other models.

For some blended learning courses, instructors may provide additional learning materials as videos or even provide video lecture on core concepts. In these circumstances, students' interaction with video resources can be extracted and used as predictors for academic performance. To this end, Matetic (2019) extracted and aggregated students' actions on video lecture in programming course. Specifically, the authors count the number of views of the video lecture for each students, and included them in the set of features alongside quiz scores, lab scores, and lecture scores. Artificial neural networks (ANNs) were employed to predict students' final course grade. The model achieved 96% accuracy in predicting course outcome. Moreover, the authors utilized Local Interpretable Model-agnostic Explanations (LIME) method to explain the model's prediction results. The results indicated that video views, although having small predictive power on final grade, can help identifying students who need additional support from the instructor.

An even finer data than interaction log that has been utilized in predicting student performance is keystroke data. This is not an unexpected data source since students who are struggling with programming exercises may have different typing patterns than those who perform well. Hence, typing patterns may be correlated to academic outcome of the students. As such, Leinonen et al. (2016) collected students' keystroke data from the IDE and exclude activities that change the code by more than one characters to investigate the predictive power of keystroke data on students' performance in Java programming course. Random forest classifier was selected over Bayesian network in predicting students' final exam performance. However, Bayesian network outperformed random forest classifier in distinguishing beginner programmers from experienced programmers. Edwards et al. (2020) extended the previous work by investigating differences in keystroke data between programming languages by collecting the keystroke data for Python programming course and Java programming course. The authors identified the most frequent character pairs, i.e., digraphs) and their average latencies. The random forest classifier was trained to predict whether the students' score will be in the top half of the class or not. The resulting model can predict students' performance with 62% for Python course and 72% for Java course. However, the predictive power of the digraphs was small to moderate. Thus, there should be further context-specific fine-tuning of the classification algorithm to improve the model performance.

While there are several attempts to extract students' behavior and underlying attributes that correlate with the academic performance, some studies proposed to treat these attributes as latent attributes that cannot be directly measured. Specifically, the item response theory (IRT) was used to model the relationship between these latent attributes and students' responses to the questions. For instance, Navrat and Tvarozek (2014) proposed the use of IRT to estimate students' latent attributes with IRT and to map the estimates with the final grade. The estimation of students' latent ability relied on the correctness of the programming exercises submitted to the online learning environment. The correctness status of every programming assignments were given to the logistic regression to estimate students' latent abilities based on the problem difficulty. Students were then ranked according to the estimated latent abilities and were matched to the final course grades. Wang et al. (2017) extended the item response theory by incorporating Bayesian knowledge tracing (BKT) in predicting student performance. Instead of lab questions where students have to submit their source code to the system, this study applied IRT on blank-filling and multiple-choice questions. The results from the IRT were used as students' initial knowledge, while BKT was employed to track students' knowledge over time through guess parameter, slip parameter, and learning rate. The proposed method was able to predict students' performance with 82% accuracy, outperforming models that use IRT and BKT separately.

Besides the aforementioned data sources that have been heavily utilized in the literature, recent studies have investigated additional data sources that could facilitate the prediction of students' performance even further. For instance, Sagala et al. (2022) incorporated extracurricular activities that the students participate with students' demographic and past academic performance. The extracurricular activity features were four binary variables indicating whether the students participate in each type of extracurricular activity or not. This study found that logistic regression achieved the best performance with 91% accuracy. Farghaly and El-Kafrawy (2023) is another study that introduced new data source in predicting students' academic performance. Instead of using prior academic achievement, the study employed a spatial rotation test and non-verbal reasoning test, taken at the beginning of the semester, to predict students' grade at the end of the semester. The spatial rotation test was adapted from the PSVT-R, which is a spatial visualization test used to measure the mental rotation of the 3D items. The non-verbal reasoning test was adapted from Kent University's non-verbal reasoning test, where participants were asked to find the next item in the series of images. Multiple linear regression was used to predict students' final outcome. The study found that predictive model performed better when no aggregations were performed on the test scores.

4.2.4.2. At-risk student identification Identifying low-performing students, i.e., at-risk students, has been an important task for instructors. Since advanced computer programming concepts often requires solid understanding of basic concepts, students who are struggle to grasp the basic will not be able to completely understand these advanced issues. By identifying those who are at risk or need assistance, instructors can provide support from the early stage to prevent confusion in the future. Basic information of students have been widely used in the literature to predict at-risk students. Singh et al. (2014) and Nabil et al. (2021) both used students' prior academic achievement in the classification. Singh et al. (2014) investigated how students' prior experience before attending the university impact their academic performance in the second year of the study. The study incorporated students' demographic, parents' information, high school academic performance, and attributes related to their study in the university such as attendance and their assignment score in the model. Naïve Bayes Classifier was used to predict whether students will pass or fail the course. While Nabil et al. (2021) also conducted similar experiment, the authors used students' grade from their first academic year instead of high school academic performance. The study focused on the pass/fail result of the data structure course because it is one of the prerequisites for other intermediate and advanced courses in computer science. The experiment compared the classification performance between traditional machine learning models, namely k-nearest neighbor (kNN), Support Vector Machine (SVM), logistic regression, and decision tree, and deep learning models. The experimented showed that deep neural network performed better than traditional models in identifying at-risk students at the early stage of the semester with 89% accuracy.

Sometimes demographic data may not be available to the researcher due to several reasons, e.g., data privacy. University administrative data such as students' course load for the semester is more accessible in some cases because it does not contain students' private information. Ochoa (2016) faced the issue of data availability and decided to use students' average grade from the previous courses and students' course load in classifying at-risk students. In this study, the author proposed a multi-level clustering model to cluster students based on these two features. An adaptation strategy was implemented to ensure that the clustering model has high enough specificity to detect at-risk students while maintain the particular size of each cluster. Brier score was used to measure the performance of the clustering model, where lower Brier score indicate more accurate prediction. The experimented showed that the adaptive multi-level clustering model performed better than the static model.

Another data source that is accessible to the instructor is exam and assignment score. Erickson (2019) collected grade from previous computer science courses, score of the first homework, and score of the first midterm exam to predict students who are at risk. The reason why this study decided to use these data is that identifying at-risk students should occur at the early stage to allow instructors to intervene early, thus the classification should rely on little prior data to make the data collection process more efficient for the instructors. The evaluation process compared the performance between linear regression and logistic regression, where linear regression will take every attributes as inputs while logistic regression will have several variants based on the number of model inputs. The experiment result showed that the logistic regression with only the score of the first homework as input is sufficient to identify at-risk students early in the semester. Similarly, Vives et al. (2024) also utilized exam scores and assignment scores alongside demographic data in predicting at-risk students. The study implemented Generative Adversarial Network (GAN) and Synthetic Monitory Oversampling Technique (SMOTE) to handle class imbalance in the dataset. The authors compared the classification performance of several traditional machine learning models and deep learning models, namely deep neural network and LSTM. Classification was done at week seven, eight, twelve, and sixteen. The experiment result showed that LSTM with GAN had the best performance with 98.3% accuracy. This study not only highlight the capabilities of early detection system but also tools for dealing with class imbalance, which is a common issue in at-risk student identification task.

Similar to the task of predicting student performance, literature has explored approaches to collect and extract student behaviors from various sources. By introducing and incorporating new data sources to the model training, it can provide instructors new understanding of the patterns of low-performing students. One example of new data sources is interaction log. Azcona and Smeaton (2017) collected programming submissions and interaction logs from the virtual learning environment

(VLE) from a 12-week course to predict whether the students will pass the next laboratory exam or not. Specifically, one classifier was trained each week, and the target is to predict the result of the upcoming exam. For classifiers from the first week to the sixth week, their target is the result of the midterm exam, while classifiers from the seventh to the twelfth week will predict the result of the final exam. After training and validating several traditional models, logistic regression was found to have the best performance. Similarly, Costa et al. (2017) also incorporated activity log in their classification algorithm, but they extended the classification to cover both distance learning and on-campus learning. Due to differences in mode of learning, different activities were extracted for distance learning and on-campus learning. Examples of activities collected in the distance learning are participation in the discussion forum, number of times the students viewing the file, and the use of educational tools. After the empirical evaluation, Support Vector Machine (SVM) was found to outperform other classifiers. Since compiling and testing are integral parts of the programming learning, Richards and Hunt (2018) analyzed students' behavior on BlueJ by tracking their actions such as editing, compiling, and testing. The authors developed the normalized programming state model (NPSM) specific to the BlueJ platform that the students used. The constructed NPSM was then evaluated on real dataset. The result showed that, while the proposed model still lack in predictive abilities, the proposed model can capture contexts specific to the BlueJ IDE.

Shrestha and Pokharel (2019) approached this issue as both clustering task and classification task. k-means clustering was used to group students into six clusters based on their number of clicks, number of quiz attempts, and number of completed activities. Instead of using binary outcome, the authors formulated the classification problem to be multi-class classification instead. Specifically, the SVM was trained to classify students into three groups, namely high, medium, and low. Similarly, Au et al. (2022) also approached this task as a multi-class classification. The study included the activity log, prior programming background, and demographic information in their classification algorithms. Two datasets were developed to evaluate the classification performance. The difference between the two datasets is the features of activity log. The features in the first dataset contain fine-grained action of the students while the features in the second dataset were aggregated. The best performing model for both datasets were the random forest. Since activity log often contains several activities recorded by the system, Buschetto Macarini et al. (2019) classified the recorded activities into four dimensions, namely cognitive, social, teaching, and other. The authors proposed that these dimensions are critical elements needed for a successful computer-assisted educational experience. The authors also experimented with 13 sets of features consisting of a combination of activities from these four dimensions in classifying at-risk students. The experiment used the data from four semesters to evaluate the performance of the classification algorithms. While the model performance varied among each dataset, the best performing model were able to detect at-risk students in the first week of the course with AUC ROC of more than 0.7.

While it is possible to extract alarming patterns from various sources of data, incorporating students perception of their own ability in the model can be beneficial. Veerasamy et al. (2021b) combined assignment and exam scores with two self-assessment surveys results, namely the problem-solving inventory questionnaire and the prior programming knowledge survey, intending to reflect students' inherent information in classifying at-risk students. The problem-solving inventory (PSI) questionnaire consists of 32 questions, measuring individual self-perception of problem-solving ability. The prior programming knowledge (PPK) survey aims to measure students' perception of their prior knowledge in programming. In the survey, the students were also asked to put the names of the programming languages that they had learned. The best performing model used the result from the PSI questionnaire and formative assessment tasks in predicting at-risk students, with 70.91% accuracy.

The issue of external validity of the classification algorithms in at-risk student identification task is prominent. Since previous studies utilized data sources that are relevant to their context or local demographics, it can be difficult to implement the findings from the literature in other environments. To address this gap, Arakawa et al. (2022) proposed the framework to identify struggling students using features that are agnostic to courses and contexts. The study collected submission information, GitHub commits for each submission, and automated testing results and extracted eight features that will be used in the classification algorithms. Since the collected data is sequential, recurrent neural network and long short-term memory were employed and evaluated. The experimented showed that wide LSTM had the best performance in identifying at-risk students, with AUC ROC of 0.922.

While previous works primarily focused on individual-level data such as assignments and self-assessments, the method proposed by Petkovic et al. (2014) investigated the activity data from group projects instead. The study analyzed the activity for the group projects on several measures such as time spent on meetings, number of email exchanges, and quality of the repository commits. The random forest classifier was employed to predict the group outcomes, identifying groups that are likely to fail. After the empirical evaluation, the authors found that group dynamics and collaboration metrics were useful in predicting failing teams in the early stage of the semester.

With the increasing discussion of explainable AI, literature. Quille et al. (2023) highlighted the importance of explainability and transparency in educational AI models and developed the Predict Student Success (PreSS) model, which designed to identify low-performing students. The proposed method used programming self-efficacy, mathematics performance, and time spent playing computer games as inputs. Decision tree was employed in this study due to its explainability. Moreover, subgroup evaluation was conducted to address algorithmic biases in the model.

4.2.4.3. Dropout prediction An extreme case of monitoring students' academic performance is to identify dropout among students. Student attrition can be measured in university level, program level, and course level. In this application, feature importance plays an important role in understanding the model result. As a result, tree-based models were used in the literature thanks to their explainability. Pereira et al. (2019) proposed a method to predict student dropout using data from the first two weeks of the semester. In this study, dropout is defined as having an attendance rate of less than 75% in the introductory programming course. Five features were extracted from the online judge and were used in predicting student dropout, namely number of logical lines for each submission, number of test cases passed, number of test cases passed for submissions with more than 50 log lines, number of student logins between the beginning and the end of a session, and keystroke latency of students. Decision tree C4.5 was used to classify dropout students from these features with 80% accuracy. The study found that students who are committed to learning, i.e., regularly solve problem sets and access the online judge system, usually complete the course. Naseem et al. (2019) explored student attrition prediction task at the program level. The study collected students' demographic data, prior education background, and financial data from the university database, and collected students' interaction with the learning management system and assessment information from Moodle. The study employed the random forest classifier to predict students who will dropout from the computer science programs during their first year of study. The classification result indicated that students' academic performance from introductory programming courses was found to be the most important predictor for dropout in computer science students.

Those students with high risk of dropout should exhibit some patterns that are different from those who intend to stay. Recent study from Triayudi et al. (2021) proposed a complete linkage dissimilarity increment distribution-global cumulative score standard (CLG), which is a combination of complete linkage (CL) algorithm, dissimilarity increment distribution (DID) algorithm, and global cumulative score standard (GCSS) algorithm, to predict dropout risk. The study collected log data from UNIX commands and source code editing during the programming classes and used them for the clustering task. After the empirical evaluation, it was found that students with minimal command inputs or sudden spikes in activity were more likely to be identified as at risk. This indicated that detecting outlier in behavioral patterns of students can be used to identify students that need further investigation or assistance from the instructor.

4.2.4.4. Knowledge tracing While exercises and exams were traditionally used to assess students' knowledge at a point in time, sometimes instructors are interested in understanding the trajectory of students' knowledge over time. Knowledge tracing refers to the estimation of students' knowledge over time using historical performance. This is a relatively new field with promising benefits to both instructors and students because it can be used to understand students' progress in the course. One of the easiest to track students' knowledge is to look at their attempts in completing the exercises since there are clear rubric in evaluating the exercises. Jiang et al. (2020) proposed an approaching index to measure the similarity between the student's intermediate submission and the perfect solution, and tracked this index over time to represent student's learning path. The approaching index is the minimum tree edit distance between student's source code and the optimal solution. The approaching index was tracked and stored in a sequence for each exercise, and was used as a feature in predicting whether the student will successfully complete the next exercise. The authors conducted empirical evaluation between logistic regression, where approaching index sequence was aggregated into one score, and recurrent neural network, where the entire approaching index sequence was used as an input. Compared to other baseline models, the proposed models performed well in classifying failing students, and were the only two models that were able to correctly identifying more than 60% of the failing students.

Kantharaju et al. (2022) conducted a knowledge tracing task on a parallel programming educational game by extracting a player's actions to predict a student's problem-solving strategy, i.e., trial and error, single-thread problem solving, and multi-thread problem-solving. The game was designed to help students learn and practice the concepts of parallel programming through visual representation. Real-time telemetry data regarding players' actions were collected throughout the level, and then divided into fixed-size time windows for feature extraction. Then, the system detect whether the player successfully apply a certain skill in the level by using the features extracted from the previous step as input to the classifier, and domain knowledge rules set by the experts. Lastly, the classification result, the domain knowledge rules, and skills required for the current level are used to update player's knowledge. Empirical evaluation was conducted to evaluate classification performance of several machine learning models. The experiment showed that, tree-based models have the best performance when the features do not contains the domain knowledge rule, while Naïve Bayes performed the best when incorporating both action features and rules.

Although previous studies explored knowledge tracing of questions or tasks that have pre-defined answers, Liu et al. (2022) expanded the knowledge tracing task to open-ended questions. Open-ended knowledge tracing (OKT) consists of knowledge representation, knowledge estimation, and response generation. Knowledge representation focuses on converting the questions and students' submissions to continuous representations. The study utilized GPT-2 to encode the questions and ASTNN to embed students' submissions. Knowledge estimation, on the other hand, estimates students' current knowledge using their past information. This study implemented a long short-term memory (LSTM) model to update students' current knowledge level by taking the concatenation of question embedding and submission embedding as input. Lastly, response generation will predict the student code based on their estimated current knowledge level. The base GPT-2 was fine-tuned into a text-to-code model to predict the student's code for the given question. In essence, OKT uses students' current knowledge extracted from previous submissions to predict their solution for the next question.

4.2.4.5. Student behavior classification Besides students' knowledge, machine learning can be used to extract students' behavior from different data sources. Student behavior cannot be simply measured by grade or attendance, since these measures are not designed to comprehensively capture student's behavior. Student's behavior can span across multiple dimensions, and extracting behavior from rich data source can provide insights that traditional metrics cannot. One data source that can be used to extract student's behavior is their interaction with the learning management system since it contains fine-grained data that can be translated to their behavior. McBroom et al. (2016) clustered students' submission activity logs from an auto-grading system into six groups and tracked the change in students' behavior over time. Features related to submissions such as percentage of early submissions, percentage of late submissions, and number of attempts in fixing compilation errors were used to cluster the students. The resulting cluster criteria were then used to track students across the semester to see if there's any evolution in the learning behavior. The proposed method was found to be beneficial in identifying common student behavior each week.

Besides submission activity, behaviors can also be extracted from how students view the learning resources. Poon (2019) extracted students' access patterns, enabling the discovery of their behaviors, which can be used as input for exam score prediction. The author utilized the hierarchical latent tree analysis, a method traditionally used for topic modeling, to capture occurrences of access to learning resources and group similar learning resources together. The study also conducted empirical evaluation of the proposed method by using the pattern tree as features in the academic performance task. The experimented showed that, by incorporating the features extracted by the proposed method, the prediction model had significantly better performance than other baseline models.

Chen et al. (2022) leveraged students' activity logs from Python MOOCs to identify and predict students' outcomes, identify factors contributing to their outcomes, and create students' profiles. The study utilized clickstream data from more than 500 students containing numerous actions such as play the video, pause the video, submit the assignment, etc. from a Python programming MOOC. Since the clickstream data is continuous, k-means clustering algorithm was used to discretized the time intervals and n-gram extraction was used to model behavioral sequences of the students. These patterns were then employed as features for predicting student academic performance. The study found that some behaviors, such as help-seeking behaviors, evaluation behaviors, and study behaviors have positive relationship with academic performance.

4.2.4.6. Learning style classification It is also possible, and sometimes necessary, for instructors to understand students' learning styles. Learning styles describe how students handle and interpret information in learning environments. Traditionally, questionnaires were employed to gather information about student's learning style. For instance, Yusoff and Najib Bin Fathi (2018) employed a VAK survey to collect students' information and incorporated it with their academic performance to classify students into three learning styles: visual, auditory, and kinaesthetic. The authors experimented with several clustering algorithms such as k-means clustering, DBSCAN, and hieararchical clustering in clustering the questionnaire results. The experimented indicated that it is possible to determine student learning styles from the data. Instead of the VAK survey, Karagiannis and Satratzemi (2019) collected student learning preference using the index of learning style (ILS) questionnaire and incorporated the results as ground truth in the classification model. Patterns of behavior were extracted from Moodle and several classifiers were trained to classify the learning style. Experiment showed that Bayesian network and decision tree can detect student's learning style earlier and more accurately than other classifiers.

Besides determining which learning styles are preferred by each student, understanding how students react can help instructors plan interventions that are appropriate to their styles. Feklistova et al. (2020) investigated how MOOC learners overcome difficulties in learning programming, analyzing data from 580 students who completed the MOOC course. pre-course and post-course questionnaires on demographics, prior experience, and actions taken when faced with challenging programming exercises. The authors implemented k-means clustering algorithm to group these results and found that demographic had little impact on student's learning strategy, while prior experience in programming and web-based education influence how students solve problems.

4.2.4.7. Other applications While grade is one of the commonly used metrics in education and research, it may not be able to fully capture student's competency in certain skills. Understanding of student's programming competency can help instructors plan and adjust their content and exercises to be appropriate with the student's competency throughout the course. Somasundaram et al. (2015) viewed this problem as a Multi-Criteria Decision Making (MCDM) problem. The study proposed a classification system that can classify students into their corresponding proficiency level in programming languages. The Analytic Hierarchy Process (AHP) was used to analyze student's interaction with the integrated development environment (IDE). The classification method takes these programming-related data as input and output the expertise level that is more relevant to each student. Another approach proposed by Silva et al. (2024) in determining student's competency is to implement clustering algorithm on student's source code. In order to cluster students based on their skill level, 23 hand-crafted features representing programming skills were defined and were used in the feature extraction process. After performing the clustering, the cluster midpoint calculation was conducted to further analyze each cluster.

Another important aspect in programming is teamwork. Usually, programmers work together on the same code base. Hence, ability to work together is crucial for every programmer. Gitinabard et al. (2023) analyzed the use of commit messages in student's GitHub repositories to investigate how students work together. Commit messages were classified into its types, e.g., merge commits, documentation commits, bug fix commits, test case commits, etc. Teamwork style classifier was trained using several features related to the commit messages such as percentage of commits for each user, length of commit messages, and total number of pair programming commits. Decision Tree with bagging was found to have the best performance in classifying teamwork style. This is the only study that utilized the GitHub's commit messages in the analysis, highlighting the opportunities to incorporate this or other alternative data source in future studies when the emphasis is on student's behavior.

It is worth noting two observations from this analysis. Firstly, there are numerous applications of AI and ML in teaching and learning computer programming. Examples of their applications include personalized learning, virtual assistant, assignment grading, feedback generation, academic performance prediction, and at-risk student identification. Secondly, there are notable advancements in the field, namely, the investigation of new data sources and the exploration of new applications. New data sources have been utilized to extract new information from the students, e.g., video (Behera et al., 2020) and log data (Chen et al., 2022). With the advancement in computational capabilities, recent literature has tackled how these new capabilities can be applied to new tasks such as developing lesson plans (Rahman & Watanobe, 2023) as well as mapping course outcomes and program outcomes (Sengupta & Das, 2023).

4.3. RQ3. Which AI and ML techniques were used in teaching computer programming courses?

This research question concerns the implementation of the applications discussed in the previous research question. In contrast to the previous research question, the focus of this research question is given to the models, algorithms, and related techniques proposed in the literature. One may view this research question as an extension of the previous research question. Now that we know how AI and ML can be used in the classroom, it is time to take a closer look at the methodologies of the studies. Since the field of AI and ML consists of several techniques, the author categorizes the techniques into three categories, namely, traditional machine learning, deep learning, and explainable AI.

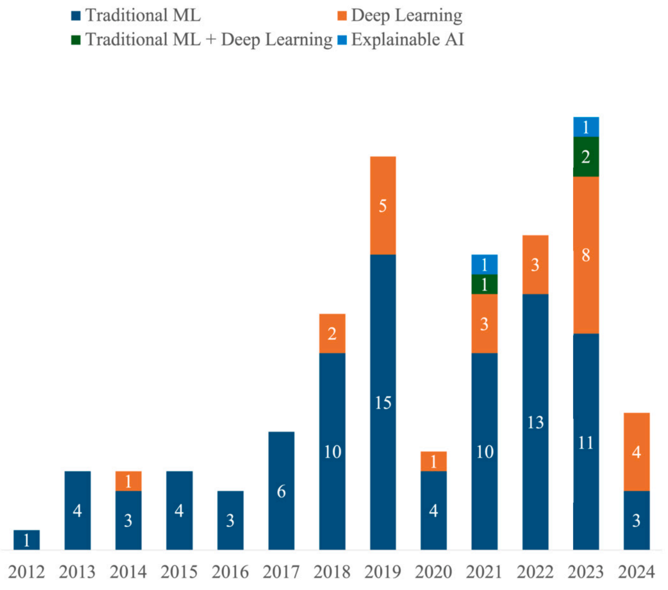

Fig. 5. Number of articles retrieved using machine learning methodology, 2012-2024.

Although several machine learning techniques have been used in the literature, it is not unexpected that recent literature utilized more advanced machine learning algorithms in their work rather than traditional algorithms. As illustrated in Fig. 5,[3] while the majority of the publications utilized traditional machine learning algorithms, there have been an increasing number of scholarly articles that adopted deep learning as their main methodology.

Before 2018, every obtained literature employed traditional machine learning algorithms as their methodology, except for Krishna Kishore et al. (2014) which employed multi-layer perceptron (MLP) to predict students' final grade. The use of deep learning models began in 2018 where deep learning model was used to predict students' academic performance (Tanuar et al., 2018) and perform knowledge tracing (Jiang et al., 2020). 2018 is also the year in which new applications were proposed in the literature, including but not limited to, feedback generation (McBroom et al., 2018) and knowledge tracing (Jiang et al., 2020). Moreover, the increase usage of traditional machine learning algorithms in 2019 came from an increase in publications regarding virtual assistant (Hobert, 2019; Saini et al., 2019) and at-risk student identification (Lobanov et al., 2019; Rubio-Manzano et al., 2019). In 2019, deep learning models, which were initially employed in academic performance prediction, were also introduced in assignment grading (Rezende Souza et al., 2019), code completion (Terada & Watanobe, 2019), and source code error detection (Gupta et al., 2019).

As discussed in section 4.1, the significant fall in the number of obtained articles in 2020 was due to the impact of COVID-19 to the research production. Nevertheless, the number of literature recovered and returned to the similar level in 2023. It is noteworthy that the use of deep learning models between 2021 and 2022 were similar to those

[3] The small number of articles in 2024 is due to our data collection being limited to the period between January 1st and April 9th, 2024.

Table 5
Traditional machine learning algorithms proposed in the literature.

Traditional Machine Learning	Number of publications
Tree-based models	22
Clustering algorithms	18
Natural language processing	14
Logistic regression	8
Naïve Bayes Classifier	5
Support Vector Machines	4
Linear regression	2
Other	16

Table 6
Deep learning algorithms proposed in the literature.

Deep Learning Technique	Number of Publications
Transformer	12
Long short-term memory	5
Multilayer perceptron	4
Convolutional neural network	4
Other	5

in 2019. That is, literature employed deep learning models in the assessment and feedback category and the performance monitoring category. However, literature in 2023 began to explore the power of deep learning models in the course design category and the classroom implementation category thanks to the introduction and popularization of large language models (LLMs). The discussion dedicated to the transformer-based models will be covered later in this subsection.

4.3.1. *Traditional machine learning*

Numerous traditional machine learning algorithms and models have been employed in the field of computer programming education. Table 5 summarizes the proposed methodologies from the retrieved articles. If the papers employed more than one algorithm, only the best algorithms are presented in this table. If the paper utilizes different machine learning algorithms on different tasks within the paper, the best-performing model for each task is presented in the table. While researchers have utilized several traditional machine learning algorithms in their studies, tree-based models, clustering algorithms, and natural language processing are most employed compared to other models. As a result, the discussion in this section will emphasize these three families of machine learning models.

Among every traditional machine learning algorithm, tree-based models were most used by researchers in numerous tasks. Specifically, tree-based models were often implemented to predict student performance (Kumar et al., 2016; Khan et al., 2019; Pereira et al., 2021a) and identify at-risk students (Buschetto Macarini et al., 2019; Ainapure et al., 2022; Quille et al., 2023). Recall that both tasks were common research topics before 2020. This signifies the importance of tree-based models as a main contributor to the research landscape before 2020. Despite the shift in the research topics after 2020, tree-based models remained relevant in the field. Beyond academic performance prediction and identification of low-performing students, tree-based models were used to perform knowledge tracing (Kantharaju et al., 2022) and identifying challenging concepts in programming course (Kerschbaumer et al., 2024). Moreover, a tree-based model has been implemented alongside transformer-based models to categorize programming problems (Pereira et al., 2021c).

On the other hand, clustering algorithms were frequently implemented in both assessment and feedback and performance monitoring categories. Clustering algorithms such as k-means and Expectation Maximization (EM) were utilized in grading programming assignments by clustering students' source code files based on program-specific features (Stankov et al., 2013; Barbosa et al., 2018; Angelovski et al., 2021; Acuna & Bansal, 2022). Researchers have employed clustering algorithms on other data sets beyond source code in other tasks. For instance, students' log data was clustered to classify their behavior patterns (McBroom et al., 2016; Chen et al., 2022). Moreover, clustering algorithms were also employed to predict student achievement (Anand et al., 2018; Khan et al., 2021) and identify at-risk students (Ochoa, 2016; Shrestha & Pokharel, 2019).

Recently, researchers have explored the use of natural language processing in classroom implementation and assessment and feedback categories. According to the search result explained in Section 3, literature on the utilization of natural language processing in computer programming education began in 2019. Compared to tree-based models and clustering algorithms, which have been the main methodologies during 2014-2016, it can be deduced that natural language processing is one of the prominent research methodologies in the contemporary research landscape. The two most common implementations of natural language processing include feedback generation (Rubio-Manzano et al., 2019; Duong et al., 2022) and virtual assistants (Hobert, 2019; Saini et al., 2019). Beyond these common applications, recent studies have explored natural language processing in assignment grading (Wong et al., 2020; Arhandi et al., 2023) as well as content personalization (Rathore et al., 2021; Huang et al., 2021).

Traditional machine learning algorithms have been utilized by past literature and are also being leveraged in recent studies. This reflected the importance of these algorithms as benchmarks for new studies. Moreover, the shift in methodology choice from tree-based models and clustering algorithms to natural language processing signifies the shift in the research landscape towards the understanding and the generation of natural language to aid the teaching and learning of computer programming.

4.3.2. *Deep learning*

With advancements in computational capabilities, researchers have been implementing several deep-learning models in the literature. This is especially true for transformer-based models such as ChatGPT and BERT. Table 6 summarizes different deep learning models proposed in the retrieved article in a similar manner to the traditional machine learning algorithms. Based on this table, transformer is the most popular methodology employed in the literature. Other deep learning models, i.e., long short-term memory (LSTM), multilayer perceptron (MLP), and convolutional neural network (CNN), have also been used in the literature.

Researchers have shown interest in grasping the power of LLMs in different research fields, and computer programming education is no exception. Tasks that traditionally cannot be performed by computers are being experimented with LLMs. For instance, the instructor generates programming questions and questions regarding computer programming concepts, which are usually done with several revisions. Sarsa et al. (2022a) proposed a method to utilize LLMs in creating programming questions and corresponding code explanations. Questions generated by LLMs such as ChatGPT were found to be of high quality by students (Speth et al., 2023). Beyond coding questions, LLMs were also experimented on generating multiple choice questions corresponding to different learning objectives (Doughty et al., 2024). Rahman and Watanobe (2023) explored the capabilities of ChatGPT in several tasks, including lesson plan development, programming question generation, and chatbot. These applications reflect the capabilities of LLMs that should be further investigated to understand their potential and limitations fully.

The use of long short-term memory (LSTM) models concentrated in the performance monitoring category. Recall that major applications in performance monitoring, i.e., academic performance prediction and at-risk students, heavily relied on tree-based models in the past. The introduction of LSTM to these topics broadens available tools and computing capabilities. Arakawa et al. (2022) attempted to use LSTM to identify at-risk students early in the semester. The resulting model achieved a mean AUROC of greater than 0.7 with just 2.43 days of data on an assignment that lasted for 14 days. Multi-task multi-layer LSTM was used to predict students' performance according to their log data from MOOCs with 92.52% accuracy (Qu et al., 2019). LSTM was also proposed to perform knowledge tracing on students' source code with an AUC-PR curve of 0.9807 (Jiang et al., 2020).

Similar to LSTM, the multilayer perceptron (MLP) model was used in performance monitoring. Matetic (2019) developed a feed-forward artificial neural network to predict students' grades based on their scores and their interaction with video lectures. While the feature for watching video lessons provided weak support on the prediction outcome, it helps to identify struggling students during the exploration process. Nabil et al. (2021) was able to identify students at risk of failing a course at an early stage of the semester with 89% accuracy. Recently, Liu et al. (2024) proposed a programming exercise recommender system that keeps track of students' behavior and knowledge and then recommends appropriate exercises for the students.

Convolutional neural network (CNN) has been applied to extract information from various sources. For instance, Rezende Souza et al. (2019) implemented a CNN to grade students' submissions with 74.9% accuracy. On the other hand, Gupta et al. (2019) developed NeuralBugLocator, a CNN-based system that can localize bugs in the source code. Besides source code, CNN was used to detect hand-over-face (HOF) gestures and facial expressions of students to identify difficult concepts students are facing with 86.87% accuracy.

The growing use of deep learning algorithms results from the advancement in computational capabilities. Deep learning models not only help researchers further explore existing problems but also broaden the research landscape by offering new computational capabilities to extract information from more complex data sources, e.g., images and videos. Moreover, the expansion of LLMs is leading to thorough investigations of tasks previously unexplored by researchers.

4.3.3. Explainable AI

Explainable AI is a recently emerging subfield of AI, referring to the ability of AI systems to provide clear and understandable explanations for their decisions or recommendations in educational settings. This is particularly important in AI-driven educational tools, such as intelligent tutoring systems or personalized learning platforms. While explainable AI was only employed in two studies, it is worth looking at them due to their distinct characteristics from the former methodologies discussed in the previous paragraphs. The goal of explainable AI is to produce outputs that can be understood and interpreted by humans. As a result, the literature employing this technique was recent. Luca and Chen (2020) proposed an explainable AI framework based on π-calculus and Hoare logic to automatically analyze students' source code, calculate grades, and provide feedback to students. Rico-Juan et al. (2023) integrated an explainable AI model to the online judge systems to provide feedback based on the submitted source code beyond the result from the unit testing.

Numerous machine learning algorithms have been proposed and explored to support the teaching and learning of computer programming courses. While traditional machine learning algorithms remained relevant in the field, researchers have been investigating the use of modern techniques such as deep learning and explainable AI in several settings. This change in methodologies is driven by the advancement in computational capabilities, allowing researchers to explore new applications and investigate new data sources that can be utilized by these machine learning algorithms.

4.4. RQ4. What benefits can AI and ML offer to both teachers and students in computer programming courses?

The fourth and last research question of this systematic review naturally follows the previous three research questions. With the growing number of research studies focusing on the utilization of AI and ML in computer programming education, the question of practical benefits that these technologies can bring to instructors and learners is becoming more crucial. If the benefits offered by these technologies are smaller than the costs of implementing them, the classroom may be better off by not investing in these technologies. Since both instructors and students are the main stakeholders in this context, the benefits should be considered from both perspectives.

4.4.1. Benefits to instructors

Instructors play an important role in computer programming education by ensuring students receive sufficient preparation for their subsequent courses and their future. With a higher number of students enrolled in programming courses, the workload for the instructor also increases. Thus, the introduction of tools that can be implemented to help the instructors achieve their course objectives will play a pivotal role in the course. Regarding the articles retrieved in this review, many applications are centered around the instructors. While several studies did not quantitatively compare the performance between the machine learning models with the instructor's performance, qualitative evaluation of the proposed models was usually included in the paper. Considering the instructor's perspective, integrating machine learning methods in the classroom provides two benefits: faster completion of mundane tasks and better focus on creative tasks.

In the context of teaching computer programming, mundane tasks do not mean tasks that provide little to no value to the course. In contrast, these tasks contribute significantly to the course's success. However, they are time-consuming and usually involve repetitive actions. Examples of such tasks are grading assignments and providing feedback. While both tasks allow instructors to assess students' knowledge thoroughly and serve as a touch point for both parties to exchange their opinions on the assignment, it takes a lot of time to carefully assess submissions from every student and give meaningful feedback. This situation is further exacerbated in introductory courses where the number of students is usually high.

Recall that assignment grading and feedback generation are the top two research topics in the assessment and feedback category; there are 13 and seven research papers on assignment grading and feedback generation, respectively. This suggests that these two topics have been common pain points of the instructors. Since these topics have been widely studied, several approaches have been proposed to help instructors perform these tasks easier. One approach to streamline these mundane tasks is to highlight relevant information to the instructor so they can spend their time efficiently. For instance, one can distinguish submissions that need further manual inspection, e.g., uncompilable programs that have a similar structure to the correct code, while disregarding submissions that do not (Angelovski et al., 2021). Automatically comparing student's submissions with the correct code is also employed in Acuna and Bansal (2022), and Arhandi et al. (2023). Another approach is to let the machine take over the task from the instructor. For instance, the automatic assessment system proposed by Ainapure et al. (2022) takes different data, e.g., unit testing results, time complexity, source code errors, etc., into consideration in the grading process. Another example of this approach is the AP-Coach developed by Duong et al. (2022), which automatically extracts the program's structure and generates corresponding feedback to the user.

By freeing up instructors' time from repetitive tasks, they can dedicate more time to creative work. The key difference between these tasks and the mundane ones mentioned earlier is that the time spent on creative tasks is used for generating ideas and finding solutions rather than carrying out repetitive actions. For example, tasks such as creating lesson plans and identifying at-risk students are considered creative because they call for instructors to carefully consider their ideas and then execute the tasks based on their ideas. Before creating their lesson plan, instructors need to first align their course objectives with the program objectives. Subsequently, they can craft their course to ensure that it aligns with the program objectives and captivates the students' interests. Similarly, instructors need to devise an effective approach to identifying underperforming students before gathering data and implementing the solution.

While some creative tasks, such as identifying at-risk students, have been studied for many years, research on other creative tasks, such as developing lesson plans, has just been recently explored thanks to the advancement of computational capabilities in deep learning algorithms and LLMs. One of the common themes in identifying at-risk students is to identify such students earlier in the course because instructors can spend more time and resources to aid those students. The nature of this problem poses a challenge to the classification model as there will be fewer data points to train the model. However, many studies have overcome this problem and can correctly identify low-performing students in the early stage of the course (Buschetto Macarini et al., 2019; Nabil et al., 2021). One example of the utilization of deep learning models on creative tasks is knowledge tracing. With deep learning, the instructor can track students' knowledge acquisition as well as predict their responses to open-ended questions (Liu et al., 2022). With the introduction of LLMs, their generative capabilities have been explored on tasks that have been traditionally manually completed by the instructor. For instance, ChatGPT has been explored in several tasks, ranging from developing lesson plans, generating questions, and acting as a chatbot with desirable performance (Rahman & Watanobe, 2023). Moreover, the process of mapping course outcomes (CO) and program outcomes (PO) can be enhanced with the help of the transformer-based models (Sengupta & Das, 2023).

4.4.2. Benefits to learners

In a classroom, if the instructor is seen as a provider, then the student is a receiver. The instructor imparts knowledge and trains students on specific skills, and students acquire this knowledge and these skills throughout the course. Due to the difficulty of programming for many students, along with a large number of students in a course, not every student will receive the necessary assistance every time. In this situation, solely expanding the instructor's capabilities may not be enough to provide the required support to every student. It is critical to offer individually tailored support to help students navigate through the course easily. The benefits that students can gain from machine learning are twofold: access to timely assistance and a personalized learning experience.

It is common for programming students to seek help on specific issues, from the syntax of the programming language to the problem-solving approach for an assignment. While students can search for resources on the internet or ask the instructor directly, they may not receive the answers they need in a timely manner. For example, students who have just started learning programming may not know how to effectively search for answers on the internet. Additionally, there is often a delay in the instructor's response. Consequently, they may remain stuck with the same problem until they can figure it out by themselves or until they finally receive the answer from the internet or the instructor. Machine learning can provide timely assistance to the student without the need for the instructor's intervention.

One of the most apparent examples of this is virtual assistants. Natural language processing techniques are heavily utilized by these assistants. For instance, Coding Tutor, a chatbot developed by Hobert (2019), can automatically assess students' source code and also answer open-ended questions. Recently, literature has used transformer models in these assistants. For example, implementing ChatGPT as a virtual peer-to-peer instruction model can enhance student performance Dos Santos and Cury (2023). Besides, virtual assistants, LLMs can be implemented to provide hints based on students' source code (Roest et al., 2023).

Different students have different levels of understanding of different topics, as well as different learning styles. Implementing the same instruction material and the same exercise may not be able to target every topic that each student wants to improve. For instance, a student may be able to complete an assignment on iterations with ease and need more difficult iterations exercises. If there is only one set of exercises, there is no room to further challenge this student. Even if this student achieves the learning objective of iterations, more advanced exercises would sharpen their conceptual understanding. The use of machine learning to tackle this issue often provides a way to offer personalized experiences to students in several ways.

Oftentimes, creating a personalized learning experience consists of several features. Rathore et al. (2021) serves as a clear example of this. The system assists students in learning computer programming and offers several features, including a material recommender and chatbot. Students are ranked based on their proficiency and will receive different study materials corresponding to their skill level. Similarly, Huang et al. (2021) designs an intelligent teaching system for Python programming. The system provides personalized learning resources as well as programming problems, nurturing independent programming thinking. Adewale et al. (2024) is another example of an intelligent teaching system. The system combines adaptive content, adaptive learning path, and context awareness of students into the system to enhance student's learning experience.

Utilizing artificial intelligence and machine learning in computer programming education is beneficial to both instructors and learners. For instructors, they can streamline mundane tasks and focus on creative tasks easily and efficiently with machine learning. For learners, they can receive timely assistance whenever they want, and they can also receive personalized learning based on their current knowledge and their areas to improve. The fact that the literature has covered these two areas reflects the equal importance of instructors and learners in the classroom. While some applications of AI and ML are centered around either instructors or learners, their impact will contribute to the course's success, whether directly or indirectly.

5. Discussion

This systematic review paper covers 119 scholarly articles published between 2012 and 2024 regarding the application of artificial intelligence (AI) and machine learning (ML) in computer programming education in higher education. The results from this systematic review provide valuable insights for both researchers and educators who would like to explore the capabilities of AI and ML in computer programming education. Firstly, AI and ML have been extensively utilized across the teaching process, indicating its capabilities for end-to-end instructor support. While the majority of literature has concentrated in the performance monitoring and assessment and feedback, the introduction and popularization of large language models (LLMs) has expanded the scope of AI applications to course design and classroom implementation, enabling and enhancing tasks such as lesson plan development, virtual assistant, and question generation. Secondly, the shift in the research landscape from performance monitoring and assessment and feedback to course design and classroom implementation reflected the growing need for personalized learning and efficient classroom management. This can be observed through the evolution of data sources from students' demographic information, which were heavily utilized by the previous studies, to fine-grained data such as LMS interaction logs and keystroke data, which were introduced in recent studies. Lastly, the AI applications covered in this systematic review do not only provide benefits to the instructors but also to the students. While the aim of this review paper is to cover AI applications from the instructors' perspectives, these applications were designed to support students' learning experience. This reflected the role of instructors in guiding their students throughout the learning journey and ensuring that they successfully meet the learning objectives of the courses.

The findings of this systematic review have significant implications for educational theories and pedagogical practices in computer programming education. Firstly, AI applications can support differentiated instruction (Suprayogi et al., 2017) by personalizing content to individual learning styles, aligning with theories of personalized learning. Moreover, these AI applications can enhance formative assessment practices by offering real-time feedback to students, which is related to the principles of learning assessment (Wiliam, 2011). Overall, AI and ML tools discussed in this systematic review align with the constructivist learning theories (Bada & Olusegun, 2015), where students iteratively construct their knowledge through interaction with learning resources and feedback. Additionally, since the AI and ML technologies enhance efficiency in several tasks, they also help reduce the cognitive burden on both instructors and students, aligning with the cognitive load theory (Navrat & Tvarozek, 2014).

The aforementioned trends highlights several opportunities for future research. Firstly, new applications of AI and ML in classroom implementation and course design categories should be further explored and investigated. As seen in Table 4, there are fewer studies on course design and classroom implementation categories than in assessment and feedback and performance monitoring categories. Therefore, it is necessary for future studies to introduce AI and ML to these categories by proposing new applications or enhancing the performance of existing solutions. For instance, with continuous and rapid development of LLMs, one may leverage the work of Rahman and Watanobe (2023) by comparing the performance between different models, or conduct an experiment to evaluate the model effectiveness. Secondly, as literature begins to cover a wide range of applications that can be implemented in the classroom for all categories, future studies should also emphasize the design of end-to-end machine pipelines that facilitate the teaching and learning computer programming from designing the course structure to evaluating students and monitoring student's performance. Since past literature often focused on a specific task or a small group of tasks, developing an end-to-end pipeline will bring state-of-the-art models from different applications together to form a complete infrastructure that can support the classroom in every aspect. Lastly, it naturally follows that this infrastructure need to be implemented and tested in the classroom to evaluate its effectiveness. These research opportunities will serve as a foundation for education institutions, researchers, and industries, in developing a computer programming course that need little to no human intervention and thus can be scaled easily.

With increasing number of literature exploring the opportunities to integrate AI and ML in computer programming education, this also poses challenges and raises one of the important questions to higher education institutions. While there is room for academic misconduct by students (Becker et al., 2023) and existing Artificial Intelligence Generated Code (AIGC) Detectors do not have desirable performance in differentiating between human-written code and AI-generated code (Pan et al., 2024), recent studies have developed a more robust model for this task (Nguyen et al., 2024; Xu & Sheng, 2024). However, the challenge for higher education institutions is to reconceptualize their roles and responsibilities where the learning environment and the education ecosystem utilize more AI and ML technologies. Since the COVID-19 pandemic, the transformation and adoption of technological tools and platforms was accelerated to accommodate online learning, for example MOOCs, web-based learning platforms and videoconferencing tools (García-Morales et al., 2021). Although the traditional model of higher education before the COVID-19 pandemic remains undisrupted by MOOCs (Al-Imarah & Shields, 2019), the MOOC market after the pandemic has been of great interest by several countries such as Bangladesh and India (Amit et al., 2022). This raises the needs for higher education institutions to revisit their positioning in the education ecosystem such that they can not only stay relevant to the rising trends of online instructions and AI but also develop new innovations with AI and ML to further enhance their teaching capabilities.

While this systematic review provides a comprehensive analysis of AI and ML applications in computer programming education, the following limitations should be considered. Firstly, the proposed search methodology, although rigorous, was limited to major databases. Thus, it is possible that the search results may excluded relevant studies from other sources. Moreover, this systematic review exclusively focused on journal and conference articles published in English between 2010 and 2024. This review does not include articles from other languages or formats such as books and theses. Secondly, the selection strategy involved manual screening of articles' titles and abstracts, potentially introducing subjectivity in the inclusion and exclusion of the studies. Thirdly, this review paper limited the scope of the review to higher education, excluding primary education and secondary education. Lastly, this article retrieved the articles up to 9 April, 2024. This retrieval period may not fully capture the rapid advancement in AI and ML technologies, especially advancements in large language models (LLMs). Since the field of AI and ML is growing rapidly, this systematic review may have not covered the most recent AI technologies in the field. Future studies should expand this review paper in terms of the review scope to cover a broader range of education level as well as more recent advancements in the field.

6. Conclusions and future directions

With the increasing demand for a computer programming workforce, artificial intelligence and machine learning technologies have been utilized in several educational and instructional tasks to enhance the teaching and learning of computer programming courses at university levels. This systematic literature review was conducted to summarize the contemporary landscape of research utilizing machine learning in this specific domain using the PRISMA protocol. The 119 papers included in this systematic review were categorized into four teaching processes, namely course design, classroom implementation, assessment and feedback, and performance monitoring. Use cases of machine learning approach in teaching computer programming courses as well as implemented machine learning methods were highlighted, focusing on benefits that machine learning technologies can bring to both instructors and learners. The main contribution of this systematic review is a methodical examination of existing research emphasizing the utilization of artificial intelligence and machine learning in teaching university-level computer programming courses. This article provides an overview for instructors looking to adopt these intelligent technologies in their classrooms and researchers seeking to develop novel machine learning models in this particular field.

CRediT authorship contribution statement

Pisut Manorat: Writing – original draft, Methodology, Visualization, Formal analysis, Data curation. **Suppawong Tuarob:** Writing – review & editing, Validation, Supervision, Project administration, Funding acquisition, Conceptualization. **Siripen Pongpaichet:** Writing – review & editing, Validation, Supervision, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Statements on open data, and ethics

The data used in this systematic literature review is extracted from publicly accessible databases and scholarly publications. All sources are appropriately acknowledged in the reference section, with detailed citations for each included study. Since this research involves a comprehensive analysis of existing literature, ethical approval and consent were not required.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research project has been funded by Mahidol University (Fundamental Fund: fiscal year 2024 by National Science Research and Innovation Fund (NSRF)), grant number FF-018/2567.

References

Acuna, R., & Bansal, A. (2022). Using programming autograder formative data to understand student growth. In *2022 IEEE frontiers in education conference (FIE)* (pp. 1–8). Uppsala, Sweden: IEEE.

Adewale, O. S., Agbonifo, O. C., Ibam, E. O., Makinde, A. I., Boyinbode, O. K., Ojokoh, B. A., Olabode, O., Omirin, M. S., & Olatunji, S. O. (2024). Design of a personalised adaptive ubiquitous learning system. *Interactive Learning Environments*, *32*, 208–228. https://doi.org/10.1080/10494820.2022.2084114.

Ainapure, B., Pise, R., & Singh, D. (2022). Machine learning based code assessment systems. In *2022 5th international conference on contemporary computing and informatics (IC3I)* (pp. 1568–1574). Uttar Pradesh, India: IEEE.

Al-Imarah, A. A., & Shields, R. (2019). MOOCs, disruptive innovation and the future of higher education: A conceptual analysis. *Innovations in Education and Teaching International*, *56*, 258–269. https://doi.org/10.1080/14703297.2018.1443828.

Alzoubi, O., Fossati, D., Eugenio, B. D., Green, N., & Chen, L. (2013). Predicting students' performance and problem solving behavior from iList log data. In *Proceedings of the 21st international conference on computers in education (ICCE 2013)*.

Amit, S., Karim, R., & Kafy, A. A. (2022). Mapping emerging massive open online course (MOOC) markets before and after COVID 19: A comparative perspective from Bangladesh and India. *Spatial Information Research*, *30*, 655–663. https://doi.org/10.1007/s41324-022-00463-4.

An, Y. (2020). A history of instructional media, instructional design, and theories. *International Journal of Technology in Education*, *4*, 1. https://doi.org/10.46328/ijte.35.

Anand, V. K., Rahiman, S. K. A., Ben George, E., & Huda, A. S. (2018). Recursive clustering technique for students' performance evaluation in programming courses. In *2018 Majan international conference (MIC)* (pp. 1–5). Muscat: IEEE.

Angelovski, D., Stankov, E., & Jovanov, M. (2021). DEMAx tool based on an improved model for semiautomatic C/C++ source code assessment. In *2021 the 6th international conference on information and education innovations* (pp. 68–73). Belgrade Serbia: ACM.

Anthony, A., & Raney, M. (2012). Bayesian network analysis of computer science grade distributions. In *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 649–654). Raleigh North Carolina USA: ACM.

Anwar, S., Butt, A. A., & Menekse, M. (2023). Utilizing automated scaffolding strategies to improve students' reflections writing process. In *2023 IEEE frontiers in education conference (FIE)* (pp. 1–5). College Station, TX, USA: IEEE.

Arakawa, K., Hao, Q., Deneke, W., Cowan, I., Wolfman, S., & Peterson, A. (2022). Early identification of student struggles at the topic level using context-agnostic features. In *Proceedings of the 53rd ACM technical symposium on computer science education* (pp. 147–153). Providence RI USA: ACM.

Arhandi, P. P., Firdausi, A. T., Andoko, B. S., & Sultan Ahmad Qum Masykuro, N. (2023). Development of SQL similarity with multiple answer keys for the automated assessment process in the SQLearn application. In *2023 sixth international conference on vocational education and electrical engineering (ICVEE)* (pp. 19–24). Surabaya, Indonesia: IEEE.

Au, T.-W., Salihin, R., & Saiful, O. (2022). Performance prediction of learning programming - machine learning approach. In *Proceedings of the 30th international conference on computers in education*.

Azcona, D., & Smeaton, A. F. (2017). Targeting at-risk students using engagement and effort predictors in an introductory computer programming course. In É. Lavoué, H. Drachsler, K. Verbert, J. Broisin, & M. Pérez-Sanagustín (Eds.), *Data driven approaches in digital education, Vol. 10474* (pp. 361–366). Cham: Springer International Publishing.

Aziz, A. A., Ismail, U. H., & Ahmad, F. (2013). *Mining Students' Academic Performance*, *53*.

Bada, D., & Olusegun, S. (2015). Constructivism learning theory: A paradigm for teaching and learning. *International Journal of Research & Method in Education*, *5*, 66–70.

Bahrehvar, M., & Moshirpour, M. (2023). Agile teaching: Automated student support and feedback generation. In *2023 IEEE frontiers in education conference (FIE)* (pp. 1–9). College Station, TX, USA: IEEE.

Baker, R. S., & Yacef, K. (2009). The state of educational data mining in 2009: A review and future visions, https://doi.org/10.5281/ZENODO.3554657.

Barbosa, A. D. A., Costa, E. D. B., & Brito, P. H. (2018). Adaptive clustering of codes for assessment in introductory programming courses. In R. Nkambou, R. Azevedo, & J. Vassileva (Eds.), *Intelligent tutoring systems, Vol. 10858* (pp. 13–22). Cham: Springer International Publishing.

Beck, P. J., Jean Mohammadi-Aragh, M., Archibald, C., Jones, B. A., & Barton, A. (2018). Real-time metacognition feedback for introductory programming using machine learning. In *2018 IEEE frontiers in education conference (FIE)* (pp. 1–5). San Jose, CA, USA: IEEE.

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming is hard - or at least it used to be: Educational opportunities and challenges of AI code generation. In *Proceedings of the 54th ACM technical symposium on computer science education V. 1* (pp. 500–506). Toronto ON Canada: ACM.

Behera, A., Matthew, P., Keidel, A., Vangorp, P., Fang, H., & Canning, S. (2020). Associating facial expressions and upper-body gestures with learning tasks for enhancing intelligent tutoring systems. *International Journal of Artificial Intelligence in Education*, *30*, 236–270. https://doi.org/10.1007/s40593-020-00195-2.

Bond, J., & Dirkin, K. (2020). What models are instructional designers using today? *Journal of Applied Instructional Design*, *9*. https://doi.org/10.51869/92jbkd.

Buschetto Macarini, L. A., Cechinel, C., Batista Machado, M. F., Faria Culmant Ramos, V., & Munoz, R. (2019). Predicting students success in blended learning—evaluating different interactions inside learning management systems. *Applied Sciences*, *9*, 5523. https://doi.org/10.3390/app9245523.

Carter, A. S., Hundhausen, C. D., & Adesope, O. (2015). The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the eleventh annual international conference on international computing education research* (pp. 141–150). Omaha Nebraska USA: ACM.

Chang, X., Wang, B., & Hui, B. (2022). Towards an automatic approach for assessing program competencies. In *LAK22: 12th international learning analytics and knowledge conference* (pp. 119–129). Online USA: ACM.

Chen, L., Chen, P., & Lin, Z. (2020). Artificial intelligence in education: A review. *IEEE Access*, *8*, 75264–75278. https://doi.org/10.1109/ACCESS.2020.2988510.

Chiheb, F., Boumahdi, F., Bouarfa, H., & Boukraa, D. (2017). Predicting students performance using decision trees: Case of an Algerian University. In *2017 international conference on mathematics and information technology (ICMIT)* (pp. 113–121). Adrar: IEEE.

Clow, D. (2013). An overview of learning analytics. *Teaching in Higher Education*, *18*, 683–695. https://doi.org/10.1080/13562517.2013.827653.

Combéfis, S. (2022). Automated code assessment for education: Review, classification and perspectives on techniques and tools. *Software*, *1*, 3–30. https://doi.org/10.3390/software1010002.

Costa, E. B., Fonseca, B., Santana, M. A., De Araújo, F. F., & Rego, J. (2017). Evaluating the effectiveness of educational data mining techniques for early prediction of students' academic failure in introductory programming courses. *Computers in Human Behavior*, *73*, 247–256. https://doi.org/10.1016/j.chb.2017.01.047.

Cretu, D. M., & Ho, Y. S. (2023). The impact of COVID-19 on educational research: A bibliometric analysis. *Sustainability*, *15*, 5219. https://doi.org/10.3390/su15065219.

Demidova, L. A., Sovietov, P. N., Andrianova, E. G., & Demidova, A. A. (2023). Anomaly detection in student activity in solving unique programming exercises: Motivated students against suspicious ones. *Data*, *8*, 129. https://doi.org/10.3390/data8080129.

Dos Santos, O. L., & Cury, D. (2023). Challenging the confirmation bias: Using ChatGPT as a virtual peer for peer instruction in computer programming education. In *2023 IEEE frontiers in education conference (FIE)* (pp. 1–7). College Station, TX, USA: IEEE.

Doughty, J., Wan, Z., Bompelli, A., Qayum, J., Wang, T., Zhang, J., Zheng, Y., Doyle, A., Sridhar, P., Agarwal, A., Bogart, C., Keylor, E., Kultur, C., Savelka, J., & Sakr, M. (2024). A comparative study of AI-generated (GPT-4) and human-crafted MCQs in programming education. In *Proceedings of the 26th Australasian computing education conference* (pp. 114–123). Sydney NSW Australia: ACM.

Duong, T. N. B., Shar, L. K., & Shankararaman, V. (2022). AP-Coach: Formative feedback generation for learning introductory programming concepts. In *2022 IEEE international conference on teaching, assessment and learning for engineering (TALE)* (pp. 323–330). Hung Hom, Hong Kong: IEEE.

Edwards, J., Leinonen, J., & Hellas, A. (2020). A study of keystroke data in two contexts: Written language and programming language influence predictability of learning outcomes. In *Proceedings of the 51st ACM technical symposium on computer science education* (pp. 413–419). Portland OR USA: ACM.

Erickson, V. (2019). Identifying at-risk computer science students early in semester utilizing data-driven models. In *2019 international conference on computational science and computational intelligence (CSCI)* (pp. 865–870). Las Vegas, NV, USA: IEEE.

Farghaly, A. A., & El-Kafrawy, P. M. (2023). Programmer performance prediction with cognitive tests: A granular approach. In *2023 international symposium on networks, computers and communications (ISNCC)* (pp. 1–6). Doha, Qatar: IEEE.

Feklistova, L., Luik, P., & Lepp, M. (2020). Clusters of programming exercises difficulties resolvers in a MOOC. In *Proceedings of the 19th European conference on E-learning, ACPI*.

Freire-Morán, M. (2023). Combining Similarity Metrics with Abstract Syntax Trees to Gain Insights into How Students Program.

García-Morales, V. J., Garrido-Moreno, A., & Martín-Rojas, R. (2021). The transformation of higher education after the COVID disruption: Emerging challenges in an online learning scenario. *Frontiers in Psychology*, *12*, Article 616059. https://doi.org/10.3389/fpsyg.2021.616059.

Gitinabard, N., Heckman, S., Lynch, C.F., Gao, Z., & Barnes, T. (2023). Analysis of Student Pair Teamwork Using GitHub Activities 15.

Guerrero-Higueras, A.M., Matellán-Olivera, V., Esteban Costales, G., Fernández-Llamas, C., Rodríguez-Sedano, F.J., & Ángel Conde, M. (2018). Model for evaluating student performance through their interaction with version control systems. CEUR Workshop Proceeding 2188, 104–112.

Gupta, R., Kanade, A., & Shevade, S. (2019). Neural Attribution for Semantic Bug-Localization in Student Programs.

Haddaway, N. R., Page, M. J., Pritchard, C. C., & McGuinness, L. A. (2022). PRISMA2020: An R package and Shiny app for producing PRISMA 2020-compliant flow diagrams, with interactivity for optimised digital transparency and open synthesis. *Campbell Systematic Reviews*, *18*, Article e1230. https://doi.org/10.1002/cl2.1230.

Heo, J., Jeong, H., Choi, D., & Lee, E. (2023). REFERENT: Transformer-based feedback generation using assignment information for programming course. In *2023 IEEE/ACM 45th international conference on software engineering: Software engineering education and training (ICSE-SEET)* (pp. 101–106). Melbourne, Australia: IEEE.

Hernández-Blanco, A., Herrera-Flores, B., Tomás, D., & Navarro-Colorado, B. (2019). A systematic review of deep learning approaches to educational data mining. *Complexity*, *2019*, 1–22. https://doi.org/10.1155/2019/1306039.

Hobert, S. (2019). Say Hello to 'Coding Tutor'! Design and Evaluation of a Chatbot-based Learning System Supporting Students to Learn to Program.

Hoq, M., Shi, Y., Leinonen, J., Babalola, D., Lynch, C., Price, T., & Akram, B. (2024). Detecting ChatGPT-generated code submissions in a CS1 course using machine learning models. In *Proceedings of the 55th ACM technical symposium on computer science education V. 1* (pp. 526–532). Portland OR USA: ACM.

Chen, H.-M., Nguyen, B.-A., Dow, C.-R., Hsueh, N.-L., & Liu, A.-C. (2022). Exploring time-related micro-behavioral patterns in a Python programming online course. *Journal of Information Science and Engineering*, *38*, Article 0002. https://doi.org/10.6688/JISE.202211_38(6).

Huang, A. Y., Lu, O. H., & Yang, S. J. (2023). Effects of artificial intelligence–enabled personalized recommendations on learners' learning engagement, motivation, and outcomes in a flipped classroom. *Computers and Education*, *194*, Article 104684. https://doi.org/10.1016/j.compedu.2022.104684.

Huang, Y., Zhao, J., Qiang, Y., Hou, T., Ren, X., & Sun, H. (2021). The reform and exploration of intelligent PYTHON language teaching. In *2021 16th international conference on computer science & education (ICCSE)* (pp. 883–888). Lancaster, United Kingdom: IEEE.

Huang, Z. (2023). YONA: An intelligence question-answering system based on artificial intelligence. In *2023 4th international conference on intelligent computing and human-computer interaction (ICHCI)* (pp. 184–189). Guangzhou, China: IEEE.

Jamjoom, M., Alabdulkreem, E., Hadjouni, M., Karim, F., & Qarh, M. (2021). Early prediction for at-risk students in an introductory programming course based on student self-efficacy. *Informatica*, *45*. https://doi.org/10.31449/inf.v45i6.3528.

Jiang, B., Wu, S., Yin, C., & Zhang, H. (2020). Knowledge tracing within single programming practice using problem-solving process data. *IEEE Transactions on Learning Technologies*, *13*, 822–832. https://doi.org/10.1109/TLT.2020.3032980.

Kadar, R., Abdul Wahab, N., Othman, J., Shamsuddin, M., & Mahlan, S. B. (2021). A study of difficulties in teaching and learning programming: A systematic literature review. *International Journal of Academic Research in Progressive Education and Development*, *10*, 591–605. https://doi.org/10.6007/IJARPED/v10-i3/11100.

Kantharaju, P., Alderfer, K., Zhu, J., Char, B., Smith, B., & Ontanon, S. (2022). Modeling player knowledge in a parallel programming educational game. *IEEE Transactions on Games*, *14*, 64–75. https://doi.org/10.1109/TG.2020.3037505.

Karagiannis, I., & Satratzemi, M. (2019). Finding an effective data mining algorithm for automatic detection of learning styles. In *Proceedings of the 18th European conference on E-learning, ACPI* (p. 36).

Kerschbaumer, D., Steinmaurer, A., & Gütl, C. (2024). A code-driven exploration of key C language concepts in a CS1 class. In M. E. Auer, & T. Tsiatsos (Eds.), *Smart mobile communication & artificial intelligence, Vol. 936* (pp. 397–408). Cham: Springer Nature Switzerland.

Keuning, H., Jeuring, J., & Heeren, B. (2016). Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM conference on innovation and technology in computer science education* (pp. 41–46). Arequipa Peru: ACM.

Khan, I., Al-Mamari, A., Al-Abdulsalam, B., Al-Abdulsalam, F., Al-Khansuri, M., Iqbal Malik, S., & Ahmad, A. R. (2021). A machine learning classification application to identify inefficient novice programmers. In H. Badioze Zaman, A. F. Smeaton, T. K. Shih, S. Velastin, T. Terutoshi, B. N. Jørgensen, H. Aris, & N. Ibrahim (Eds.), *Advances in visual informatics, Vol. 13051* (pp. 423–434). Cham: Springer International Publishing.

Khan, I., Al Sadiri, A., Ahmad, A. R., & Jabeur, N. (2019). Tracking student performance in introductory programming by means of machine learning. In *2019 4th MEC international conference on big data and smart city (ICBDSC)* (pp. 1–6). Muscat, Oman: IEEE.

Koutcheme, C., Sarsa, S., Leinonen, J., Haaranen, L., & Hellas, A. (2023). Evaluating distance measures for program repair. In *Proceedings of the 2023 ACM conference on international computing education research V.1* (pp. 495–507). Chicago IL USA: ACM.

Krishna Kishore, K. V., Venkatramaphanikumar, S., & Alekhya, S. (2014). Prediction of student academic progression: A case study on Vignan University. In *2014 international conference on computer communication and informatics* (pp. 1–6). Coimbatore, India: IEEE.

Krouska, A., Troussas, C., & Sgouropoulou, C. (2020). Applying genetic algorithms for recommending adequate competitors in mobile game-based learning environments. In V. Kumar, & C. Troussas (Eds.), *Intelligent tutoring systems, Vol. 12149* (pp. 196–204). Cham: Springer International Publishing.

Kumar, S. C., Chowdary, E. D., Venkatramaphanikumar, S., & Kishore, K. K. (2016). M5P model tree in predicting student performance: A case study. In *2016 IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)* (pp. 1103–1107). Bangalore, India: IEEE.

Kurilovas, E. (2019). Advanced machine learning approaches to personalise learning: Learning analytics and decision making. *Behaviour & Information Technology*, *38*, 410–421. https://doi.org/10.1080/0144929X.2018.1539517.

Kuruppu, T., Tharmaseelan, J., Silva, C., Arachchillage, U., Manathunga, K., Reyal, S., Kodagoda, N., & Jayalath, T. (2021). Source code based approaches to automate marking in programming assignments. In *Proceedings of the 13th international conference on computer supported education, SCITEPRESS - science and technology publications, online streaming, — select a country* (pp. 291–298).

Lavbič, D., Matek, T., & Zrnec, A. (2017). Recommender system for learning SQL using hints. *Interactive Learning Environments*, *25*, 1048–1064. https://doi.org/10.1080/10494820.2016.1244084.

Leinonen, J., Longi, K., Klami, A., & Vihavainen, A. (2016). Automatic inference of programming performance and experience from typing patterns. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 132–137). Memphis Tennessee USA: ACM.

Li, M., Ge, M., Zhao, H., & An, Z. (2022). Modeling and analysis of Learners' emotions and behaviors based on online forum texts. *Computational Intelligence and Neuroscience*, *2022*, 1–14. https://doi.org/10.1155/2022/9696422.

Liao, S. N., Zingaro, D., Thai, K., Alvarado, C., Griswold, W. G., & Porter, L. (2019). A robust machine learning technique to predict low-performing students. *ACM Transactions on Computing Education*, *19*, 1–19. https://doi.org/10.1145/3277569.

Liu, N., Wang, Z., Baraniuk, R., & Lan, A. (2022). Open-ended knowledge tracing for computer science education. In *Proceedings of the 2022 conference on empirical methods in natural language processing, association for computational linguistics, Abu Dhabi, United Arab Emirates* (pp. 3849–3862).

Liu, X., Castellanos, H., Wiese, L., & Magana, A. J. (2023). Exploring machine learning methods to identify patterns in students' solutions to programming assignments. In *2023 IEEE frontiers in education conference (FIE)* (pp. 1–6). College Station, TX, USA: IEEE.

Liu, Y., Zhu, R., & Gao, M. (2024). Personalized programming guidance based on deep programming learning style capturing. In W. Hong, & G. Kanaparan (Eds.), *Computer science and education. Computer science and technology, Vol. 2023* (pp. 214–231). Singapore: Springer Nature Singapore.

Lobanov, A., Bryksin, T., & Shpilman, A. (2019). Automatic classification of error types in solutions to programming assignments at online learning platform. In S. Isotani, E. Millán, A. Ogan, P. Hastings, B. McLaren, & R. Luckin (Eds.), *Artificial intelligence in education, Vol. 11626* (pp. 174–178). Cham: Springer International Publishing.

Longi, K., Leinonen, J., Nygren, H., Salmi, J., Klami, A., & Vihavainen, A. (2015). Identification of programmers from typing patterns. In *Proceedings of the 15th koli calling conference on computing education research* (pp. 60–67). Koli, Finland: ACM.

Luca, G. D., & Chen, Y. (2020). Explainable AI for workflow verification in VIPLE. *Journal of Artificial Intelligence and Technology*, *1*, 21–27. https://doi.org/10.37965/jait.2020.0023.

Malik, A., Wu, M., Vasavada, V., Song, J., Coots, M., Mitchell, J., Goodman, N., & Piech, C. (2019). Generative grading: Near human-level accuracy for automated feedback on richly structured problems, https://doi.org/10.48550/ARXIV.1905.09916.

Matetic, M. (2019). Mining learning management system data using interpretable neural networks. In *2019 42nd international convention on information and communication technology, electronics and microelectronics (MIPRO)* (pp. 1282–1287). Opatija, Croatia: IEEE.

Matsrostefano, S., & Sciarrone, F. (2022). Monitoring programming styles in massive open online courses using source embedding. In *2022 26th international conference information visualisation (IV)* (pp. 245–250). Vienna, Austria: IEEE.

McBroom, J., Jeffries, B., Koprinska, I., & Yacef, K. (2016). Mining behaviours of students in autograding submission system logs. In *Proceedings of the 9th international conference on educational data mining*.

McBroom, J., Yacef, K., Koprinska, I., & Curran, J. R. (2018). A data-driven method for helping teachers improve feedback in computer programming automated tutors. In C. Penstein Rosé, R. Martínez-Maldonado, H. U. Hoppe, R. Luckin, M. Mavrikis, K. Porayska-Pomsta, B. McLaren, & B. Du Boulay (Eds.), *Artificial intelligence in education, Vol. 10947* (pp. 324–337). Cham: Springer International Publishing.

Mehmood, E., Abid, A., Farooq, M. S., & Nawaz, N. A. (2020). Curriculum, teaching and learning, and assessments for introductory programming course. *IEEE Access*, *8*, 125961–125981. https://doi.org/10.1109/ACCESS.2020.3008321.

Možina, M., Lazar, T., & Bratko, I. (2018). Identifying typical approaches and errors in Prolog programming with argument-based machine learning. *Expert Systems with Applications*, *112*, 110–124. https://doi.org/10.1016/j.eswa.2018.06.029.

Nabil, A., Seyam, M., & Abou-Elfetouh, A. (2021). Prediction of students' academic performance based on courses' grades using deep neural networks. *IEEE Access*, *9*, 140731–140746. https://doi.org/10.1109/ACCESS.2021.3119596.

Naseem, M., Chaudhary, K., Sharma, B., & Lal, A. G. (2019). Using ensemble decision tree model to predict student dropout in computing science. In *2019 IEEE Asia-Pacific conference on computer science and data engineering (CSDE)* (pp. 1–8). Melbourne, Australia: IEEE.

Navrat, P., & Tvarozek, J. (2014). Online programming exercises for summative assessment in university courses. In *Proceedings of the 15th international conference on computer systems and technologies* (pp. 341–348). Ruse Bulgaria: ACM.

Nguyen, P. T., Di Rocco, J., Di Sipio, C., Rubei, R., Di Ruscio, D., & Di Penta, M. (2024). GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT. *The Journal of Systems and Software*, *214*, Article 112059. https://doi.org/10.1016/j.jss.2024.112059.

Ochoa, X. (2016). Adaptive multilevel clustering model for the prediction of academic risk. In *2016 XI Latin American conference on learning objects and technology (LACLO)* (pp. 1–8). San Carlos, Alajuela, Costa Rica: IEEE.

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., … Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, *71*. https://doi.org/10.1136/bmj.n71.

Pan, W. H., Chok, M. J., Wong, J. L. S., Shin, Y. X., Poon, Y. S., Yang, Z., Chong, C. Y., Lo, D., & Lim, M. K. (2024). Assessing AI detectors in identifying AI-generated code: Implications for education. In *Proceedings of the 46th international conference on software engineering: Software engineering education and training* (pp. 1–11). Lisbon Portugal: ACM.

Pereira, F. D., Fonseca, S. C., Oliveira, E. H. T., Cristea, A. I., Bellhauser, H., Rodrigues, L., Oliveira, D. B. F., Isotani, S., & Carvalho, L. S. G. (2021a). Explaining individual and collective programming students' behavior by interpreting a black-box predictive model. *IEEE Access*, *9*, 117097–117119. https://doi.org/10.1109/ACCESS.2021.3105956.

Pereira, F. D., Junior, H. B. F., Rodriguez, L., Toda, A., Oliveira, E. H. T., Cristea, A. I., Oliveira, D. B. F., Carvalho, L. S. G., Fonseca, S. C., Alamri, A., & Isotani, S. (2021b). A recommender system based on effort: Towards minimising negative affects and maximising achievement in CS1 learning. In A. I. Cristea, & C. Troussas (Eds.), *Intelligent tutoring systems, Vol. 12677* (pp. 466–480). Cham: Springer International Publishing.

Pereira, F. D., Oliveira, E., Cristea, A., Fernandes, D., Silva, L., Aguiar, G., Alamri, A., & Alshehri, M. (2019). Early dropout prediction for programming courses supported by online judges. In S. Isotani, E. Millán, A. Ogan, P. Hastings, B. McLaren, & R. Luckin (Eds.), *Artificial intelligence in education, Vol. 11626* (pp. 67–72). Cham: Springer International Publishing.

Pereira, F. D., Oliveira, E., Rodrigues, L., Cabral, L., Oliveira, D., Carvalho, L., Gasevic, D., Cristea, A., Dermeval, D., & Mello, R. F. (2023). Evaluation of a hybrid AI-human recommender for CS1 instructors in a real educational scenario. In O. Viberg, I. Jivet, P. J. Muñoz-Merino, M. Perifanou, & T. Papathoma (Eds.), *Responsive and sustainable educational futures, Vol. 14200* (pp. 308–323). Cham: Springer Nature Switzerland.

Pereira, F. D., Pires, F., Fonseca, S. C., Oliveira, E. H. T., Carvalho, L. S. G., Oliveira, D. B. F., & Cristea, A. I. (2021c). Towards a human-AI hybrid system for categorising programming problems. In *Proceedings of the 52nd ACM technical symposium on computer science education* (pp. 94–100). virtual event USA: ACM.

Petkovic, D., Sosnick-Perez, M., Huang, S., Todtenhoefer, R., Okada, K., Arora, S., Sreenivasen, R., Flores, L., & Dubey, S. (2014). SETAP: Software engineering teamwork assessment and prediction using machine learning. In *2014 IEEE frontiers in education conference (FIE) proceedings* (pp. 1–8). Madrid, Spain: IEEE.

Plaza, L., Araujo, L., López-Ostenero, F., & Martínez-Romo, J. (2023). Automatic recommendation of forum threads and reinforcement activities in a data structure and programming course. *Applied System Innovation*, *6*, 83. https://doi.org/10.3390/asi6050083.

Poon, L. K. M. (2019). Extracting access patterns with hierarchical latent tree analysis: An empirical study on an undergraduate programming course. In H. Seki, C. H. Nguyen, V. N. Huynh, & M. Inuiguchi (Eds.), *Integrated uncertainty in knowledge modelling and decision making, Vol. 11471* (pp. 380–392). Cham: Springer International Publishing.

Qin, Y., Sun, G., Li, J., Hu, T., & He, Y. (2021). SCG_FBS: A code grading model for students' program in programming education. In *2021 13th international conference on machine learning and computing* (pp. 210–216). Shenzhen China: ACM.

Qu, S., Li, K., Wu, B., Zhang, X., & Zhu, K. (2019). Predicting student performance and deficiency in mastering knowledge points in MOOCs using multi-task learning. *Entropy*, *21*, 1216. https://doi.org/10.3390/e21121216.

Quille, K., Vidal-Meliá, L., Nolan, K., & Mooney, A. (2023). Evolving towards a trustworthy AIEd model to predict at risk students in introductory programming courses. In *Proceedings of the 2023 conference on human centered artificial intelligence: Education and practice* (pp. 22–28). Dublin Ireland: ACM.

Rahman, M. M., & Watanobe, Y. (2023). ChatGPT for education and research: Opportunities, threats, and strategies. *Applied Sciences*, *13*, 5783. https://doi.org/10.3390/app13095783.

Rahman, M. M., Watanobe, Y., Rage, U. K., & Nakamura, K. (2021). A novel rule-based online judge recommender system to promote computer programming education. In H. Fujita, A. Selamat, J. C. W. Lin, & M. Ali (Eds.), *Advances and trends in artificial intelligence. From theory to practice, Vol. 12799* (pp. 15–27). Cham: Springer International Publishing.

Raju, A. D., Gaayathre, C., Deepthi, G. L., Madhuri, K., & Maheswari, D. (2019). Prediction of students' performance for a multi class problem using naïve Bayes classifier. *International Journal of Innovative Technology and Exploring Engineering*, *8*.

Rathore, A. S., Sharma, A., & Massoudi, M. (2021). Personalized engineering education model based on artificial intelligence for learning programming. In *2021 6th international conference on computing, communication and security (ICCCS)* (pp. 1–10). Las Vegas, NV, USA: IEEE.

Rezende Souza, F., Zampirolli, F., & Kobayashi, G. (2019). Convolutional neural network applied to code assignment grading. In *Proceedings of the 11th international conference on computer supported education, SCITEPRESS* (pp. 62–69). Heraklion, Crete, Greece: Science and Technology Publications.

Richards, B., & Hunt, A. (2018). Investigating the applicability of the normalized programming state model to BlueJ programmers. In *Proceedings of the 18th koli calling international conference on computing education research* (pp. 1–10). Koli, Finland: ACM.

Rico-Juan, J. R., Sánchez-Cartagena, V. M., Valero-Mas, J. J., & Gallego, A. J. (2023). Identifying student profiles within online judge systems using explainable artificial intelligence. *IEEE Transactions on Learning Technologies*, *16*, 955–969. https://doi.org/10.1109/TLT.2023.3239110.

Roest, L., Keuning, H., & Jeuring, J. (2023). Next-step hint generation for introductory programming using large language models. Retrieved from arXiv:2312.10055.

Rubio-Manzano, C., Lermanda Senoceaín, T., Martinez-Araneda, C., Vidal-Castro, C., & Segura-Navarrete, A. (2019). Fuzzy linguistic descriptions for execution trace comprehension and their application in an introductory course in artificial intelligence. *Journal of Intelligent & Fuzzy Systems*, *37*, 8397–8415. https://doi.org/10.3233/JIFS-190935.

Sagala, T. M. N., Permai, S. D., Gunawan, A. A. S., Barus, R. O., & Meriko, C. (2022). Predicting computer science student's performance using logistic regression. In *2022 5th international seminar on research of information technology and intelligent systems (ISRITI)* (pp. 817–821). Yogyakarta, Indonesia: IEEE.

Saini, R., Mussbacher, G., Guo, J. L., & Kienzle, J. (2019). Teaching modelling literacy: An artificial intelligence approach. In *2019 ACM/IEEE 22nd international conference on model driven engineering languages and systems companion (MODELS-C)* (pp. 714–719). Munich, Germany: IEEE.

Saoban, C., & Rimcharoen, S. (2019). Identifying an original copy of the source codes in programming assignments. In *2019 16th international joint conference on computer science and software engineering (JCSSE)* (pp. 271–276). Chonburi, Thailand: IEEE.

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022a). Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM conference on international computing education research - volume 1, ACM, lugano and virtual event Switzerland* (pp. 27–43).

Sarsa, S., Leinonen, J., Koutcheme, C., & Hellas, A. (2022b). Speeding up automated assessment of programming exercises. In *Proceedings of the 2022 conference on United Kingdom & Ireland computing education research* (pp. 1–7). Dublin Ireland: ACM.

Sengupta, S., & Das, A. K. (2023). Automated mapping of course outcomes to program outcomes using natural language processing and machine learning. In *2023 IEEE 3rd applied signal processing conference (ASPCON)* (pp. 44–48). India: IEEE.

Sheshadri, A., Gitinabard, N., Lynch, C. F., Barnes, T., & Heckman, S. (2018). Predicting student performance based on online study habits: A study of blended courses. In *Proceedings of the 11th international conference on educational data mining*.

Shrestha, S., & Pokharel, M. (2019). Machine learning algorithm in educational data. In *2019 artificial intelligence for transforming business and society (AITB)* (pp. 1–11). Kathmandu, Nepal: IEEE.

Silva, D. B., Carvalho, D. R., & Silla, C. N. (2024). A clustering-based computational model to group students with similar programming skills from automatic source code analysis using novel features. *IEEE Transactions on Learning Technologies*, *17*, 428–444. https://doi.org/10.1109/TLT.2023.3273926.

Singh, G., Srikant, S., & Aggarwal, V. (2016). Question independent grading using machine learning: The case of computer program grading. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 263–272). San Francisco California USA: ACM.

Singh, M., Singh, J., & Rawal, A. (2014). Feature extraction model to identify at – risk level of students in academia. In *2014 international conference on information technology* (pp. 221–227). Bhubaneswar, India: IEEE.

Skalka, J., & Drlik, M. (2020). Automated assessment and microlearning units as predictors of at-risk students and students' outcomes in the introductory programming courses. *Applied Sciences*, *10*, 4566. https://doi.org/10.3390/app10134566.

Somasundaram, T. S., Kiruthika, U., Gowsalya, A., Hemalatha, A., & Philips, A. (2015). Determination of competency of programmers by classification and ranking using AHP. In *2015 IEEE international conference on electro/information technology (EIT)* (pp. 194–200). Dekalb, IL, USA: IEEE.

Speth, S., Meißner, N., & Becker, S. (2023). Investigating the use of AI-generated exercises for beginner and intermediate programming courses: A ChatGPT case study. In *2023 IEEE 35th international conference on software engineering education and training (CSEE&T)* (pp. 142–146). Tokyo, Japan: IEEE.

Srikant, S., & Aggarwal, V. (2014). A system to grade computer programming skills using machine learning. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1887–1896). New York New York USA: ACM.

Stankov, E., Jovanov, M., Madevska Bogdanova, A., & Gusev, M. (2013). A new model for semiautomatic student source code assessment. *International Journal of Computing and Information Technology*, *21*, 185. https://doi.org/10.2498/cit.1002193.

Stefaniak, J., & Xu, M. (2020). An examination of the systemic reach of instructional design models: A systematic review. *TechTrends*, *64*, 710–719. https://doi.org/10.1007/s11528-020-00539-8.

Suprayogi, M. N., Valcke, M., & Godwin, R. (2017). Teachers and their implementation of differentiated instruction in the classroom. *Teaching and Teacher Education*, *67*, 291–301. https://doi.org/10.1016/j.tate.2017.06.020.

Takhar, R., & Aggarwal, V. (2019). Grading uncompilable programs. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 9389–9396. https://doi.org/10.1609/aaai.v33i01.33019389.

Tanuar, E., Heryadi, Y., Lukas, A. B. S., & Gaol, F. L. (2018). Using machine learning techniques to earlier predict student's performance. In *2018 Indonesian association for pattern recognition international conference (INAPR)* (pp. 85–89). Jakarta, Indonesia: IEEE.

Terada, K., & Watanobe, Y. (2019). Code completion for programming education based on recurrent neural network. In *2019 IEEE 11th international workshop on computational intelligence and applications (IWCIA)* (pp. 109–114). Hiroshima, Japan: IEEE.

Tharmaseelan, J., Manathunga, K., Reyal, S., Kasthurirathna, D., & Thurairasa, T. (2021). Revisit of automated marking techniques for programming assignments. In *2021 IEEE global engineering education conference (EDUCON)* (pp. 650–657). Vienna, Austria: IEEE.

Thomas, A., Stopera, T., Frank-Bolton, P., & Simha, R. (2019). Stochastic tree-based generation of program-tracing practice questions. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 91–97). Minneapolis MN, USA: ACM.

Triayudi, A., Widyarto, W. O., Kamelia, L., Iksal, I., & Sumiati, S. (2021). CLG clustering for dropout prediction using log-data clustering method. *IAES International Journal of Artificial Intelligence*, *10*, 764. https://doi.org/10.11591/ijai.v10.i3.pp764-770.

Troussas, C., Krouska, A., Tselenti, P., Kardaras, D. K., & Barbounaki, S. (2023). Enhancing personalized educational content recommendation through cosine similarity-based knowledge graphs and contextual signals. *Information*, *14*, 505. https://doi.org/10.3390/info14090505.

Uchiyama, S., Kubo, A., Washizaki, H., & Fukazawa, Y. (2014). Detecting design patterns in object-oriented program source code by using metrics and machine learning. *Journal of Software Engineering and Applications*, *07*, 983–998. https://doi.org/10.4236/jsea.2014.712086.

Ulloa-Cazarez, R. L., López-Martín, C., Abran, A., & Yáñez-Márquez, C. (2018). Prediction of online students performance by means of genetic programming. *Applied Artificial Intelligence*, *32*, 858–881. https://doi.org/10.1080/08839514.2018.1508839.

Veerasamy, A. K., Laakso, M. J., & D'Souza, D. (2021a). Formative assessment tasks as indicators of student engagement for predicting at-risk students in programming courses. *Informatics in Education*. https://doi.org/10.15388/infedu.2022.15.

Veerasamy, A. K., Laakso, M. J., D'Souza, D., & Salakoski, T. (2021b). Predictive models as early warning systems: A Bayesian classification model to identify at-risk students of programming. In K. Arai (Ed.), *Intelligent computing, Vol. 284* (pp. 174–195). Cham: Springer International Publishing.

Verma, A., Udhayanan, P., Shankar, R. M., Kn, N., & Chakrabarti, S. K. (2021). Source-code similarity measurement: Syntax tree fingerprinting for automated evaluation. In *The first international conference on AI-ML-systems* (pp. 1–7). Bangalore India: ACM.

Vives, L., Cabezas, I., Vives, J. C., Reyes, N. G., Aquino, J., Cóndor, J. B., & Altamirano, S. F. S. (2024). Prediction of students' academic performance in the programming fundamentals course using long short-term memory neural networks. *IEEE Access*, *12*, 5882–5898. https://doi.org/10.1109/ACCESS.2024.3350169.

Vujošević-Janičić, M., Nikolić, M., Tošić, D., & Kuncak, V. (2013). Software verification and graph similarity for automated evaluation of students' assignments. *Information and Software Technology*, *55*, 1004–1016. https://doi.org/10.1016/j.infsof.2012.12.005.

Wang, F. H. (2023). Efficient generation of text feedback in object-oriented programming education using cached performer revision. *Machine Learning with Applications*, *13*, Article 100481. https://doi.org/10.1016/j.mlwa.2023.100481.

Wang, S., Han, Y., Wu, W., & Hu, Z. (2017). Modeling student learning outcomes in studying programming language course. In *2017 seventh international conference on information science and technology (ICIST)* (pp. 263–270). Da Nang, Vietnam: IEEE.

Wiliam, D. (2011). What is assessment for learning? *Studies in Educational Evaluation*, *37*, 3–14. https://doi.org/10.1016/j.stueduc.2011.03.001.

Wong, T. L., Poon, C. K., Tang, C. M., Yu, Y. T., & Lee, V. C. S. (2020). Automatic generation of matching rules for programming exercise assessment. In L. K. Lee, L. H. U, F. L. Wang, S. K. S. Cheung, O. Au, & K. C. Li (Eds.), *Technology in education. Innovations for online teaching and learning, Vol. 1302* (pp. 126–135). Singapore: Springer Singapore.

Wu, H., Fa, D., Wu, X., Tan, W., Chang, X., Gao, Y., & Weng, J. (2023). Research on the construction of intelligent programming platform based on AI-generated content. In *The 15th international conference on education technology and computers* (pp. 9–15). Barcelona Spain: ACM.

Xu, Y., Liu, X., Cao, X., Huang, C., Liu, E., Qian, S., Liu, X., Wu, Y., Dong, F., Qiu, C. W., Qiu, J., Hua, K., Su, W., Wu, J., Xu, H., Han, Y., Fu, C., Yin, Z., Liu, M., … Zhang, J. (2021). Artificial intelligence: A powerful paradigm for scientific research. *The Innovation*, *2*, Article 100179. https://doi.org/10.1016/j.xinn.2021.100179.

Xu, Z., & Sheng, V. S. (2024). Detecting AI-generated code assignments using perplexity of large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*, 23155–23162. https://doi.org/10.1609/aaai.v38i21.30361.

Yasaswi, J., Kailash, S., Chilupuri, A., Purini, S., & Jawahar, C. V. (2017). Unsupervised learning based approach for plagiarism detection in programming assignments. In *Proceedings of the 10th innovations in software engineering conference* (pp. 117–121). Jaipur India: ACM.

Yoo, J., & Kim, J. (2014). Can online discussion participation predict group project performance? Investigating the roles of linguistic features and participation patterns. *International Journal of Artificial Intelligence in Education*, *24*, 8–32. https://doi.org/10.1007/s40593-013-0010-8.

Yoshimura, R., Sakamoto, K., Washizaki, H., & Fukazawa, Y. (2022). WOJR: A recommendation system for providing similar problems to programming assignments. *Applied System Innovation*, *5*, 53. https://doi.org/10.3390/asi5030053.

Yusoff, M., & Najib Bin Fathi, M. (2018). Evaluation of clustering methods for student learning style based neuro linguistic programming. *International Journal of Engineering and Technology*, *7*, 63. https://doi.org/10.14419/ijet.v7i3.15.17408.

Zhao, X. (2022). Research on application of personalized recommendation technology in adaptive learning system based on Java programming. In *2022 international conference on automation, robotics and computer engineering (ICARCE)* (pp. 1–4). Wuhan, China: IEEE.