Examining the Effect of Machine-Learning Programming Simulator on Student Performance and Student Anxiety

Tansa Trisna Astono Putri[1], Wan Ahmad Jaafar Wan Yahaya[2,*], Nur Azlina Mohamed Mokmin[2], and Sriadhi Sriadhi[1]

[1]Information Technology and Computer Education Study Program of Universitas Negeri Medan, Medan, Indonesia
[2]Centre for Instructional Technology and Multimedia of Universiti Sains Malaysia, Penang, Malaysia
Email: tansatrisna@unimed.ac.id (T.T.A.P.); wajwy@usm.my (W.A.J.W.Y.); nurazlina@usm.my (N.A.M.M.); sriadhi@unimed.ac.id (S.S.)
*Corresponding author
Manuscript received December 20, 2024; revised February 6, 2025; accepted March 13, 2025; published July 18, 2025

***Abstract*—Acquiring programming skills can be a complex and daunting challenge for novice university students. Mastering the syntax of programming languages is not just a superficial endeavor; it requires students to develop a robust set of principles to tackle specific problem scenarios. Machine learning technology has the potential to be beneficial across various industries; however, its application in educational tools remains inadequate. Therefore, this project aims to implement machine learning technology in a simulator designed to assist students in evaluating their programming courses. This study developed a machine-learning programming simulator and explored its impact on students with varying levels of anxiety. Educational Data Mining (EDM) refers to the application of data mining techniques to extract valuable information and insights from extensive data repositories within the education sector. The primary goal of this approach is to evaluate student performance in programming courses. To assess the effects of technology on academic performance, the study employed Analysis of Variance (ANOVA) and Analysis of Covariance (ANCOVA) methodologies. The findings suggest that both students with high levels of anxiety and those with low levels of anxiety benefit from exposure to machine learning technology. By utilizing a machine learning programming simulator, students' performance in programming courses can significantly improve, irrespective of their anxiety levels.**

***Keywords*—programming, university students, performance, anxiety, machine-learning simulator**

I. Introduction

Programming abilities are becoming increasingly important for students in today's world. Skills in programming are highly sought after across many fields, including software development, finance, healthcare, and even the entertainment industry. Students majoring in computer programming have access to a wide range of job opportunities. To find effective solutions, they must utilize both critical and creative thinking skills. Furthermore, these abilities are useful beyond the realm of computer science, impacting many other areas of life.

As technology continues to advance and permeate every aspect of our lives, programming skills will remain in high demand. Learning to code today can help students secure future employment. Computational thinking, which involves breaking down complex issues into simpler components, is a skill that students can develop through programming education. This competency can also benefit various other fields and areas of life.

Students who enroll in programming classes may discover a new avenue for creative expression by developing original software, video games, and websites. Engaging in coding can enhance their creativity and problem-solving abilities.

According to the US Bureau of Labor Statistics, a 10% decline in the employment of computer programmers is projected from 2021 to 2031 [1]. Nevertheless, they anticipate that around 9600 new programming jobs will emerge annually, largely due to professionals leaving the field for other opportunities or retiring. While programming roles are diminishing, automation is a significant contributing factor. Many individuals whose jobs were once reliant on performing simple or repetitive tasks are finding themselves increasingly unemployed as companies automate these activities.

On the flip side, this shift creates new opportunities for individuals with diverse skill sets. Although more complex tasks may become automated, strategic positions will continue to be in high demand. To remain competitive in this evolving landscape, programmers must continually refine their skills to effectively tackle these challenging roles.

Technology Assessment and Application Agency of Indonesia [2] indicates that Indonesia needs around 600,000 programmers in 2025. Currently, the number of available programmers in Indonesia is around 100,000 individuals. This situation indicates that the country is significantly far from reaching its established goals. As a result, decisive measures are required to address the human resources dilemma in the programming sector. More supervision during lecture sessions is essential to help students develop strong programming skills and competency. The number of highly educated programmers graduating from higher education institutions is expected to rise as a result of the increased integration of technology into programming courses.

Moreover, a growing number of companies have adopted information systems as a means to enhance their operations. To streamline their processes, modern businesses have integrated these systems into their workflows. As a result, there is high demand for programmers whenever companies seek to implement information systems. Unfortunately, the number of programmers in Indonesia remains low. Therefore, it is essential to provide additional guidance during lectures to cultivate competent programmers.

The incorporation of technology in programming classes is expected to increase the number of skilled programmers graduating from universities. Various technological approaches, such as gamification, assessment tools, and visualization, can also enhance the learning experience in

programming courses.

Undergraduate computer science courses equip students with a solid foundation in the field. After graduation, students can apply this knowledge in their professional endeavors. However, those who lack proficiency in computer programming principles may face challenges in securing employment, particularly in development and software engineering roles. Consequently, programming is a mandatory subject for computer science majors. All computer science majors are now expected to have a basic understanding of computing, as well as foundational skills in reading, writing, and basic math [3]. To thrive as computer science majors in university and keep pace with their professors and course materials, students must develop certain skills.

Research published in 2016 found that only a small percentage of university students achieve academic success, while a staggering 65% either fail the course or choose not to continue. Globally, over 30% of computer science undergraduates either fail to complete the course or do not pass the programming component [4]. The diploma was canceled because the introductory program was not released [5]. Institutions, educators, and students alike must contend with the alarmingly high rates of failure and dropout. The reluctance of students to attend required classes, a core part of computer science programs, is problematic as it often leads to failure and delayed graduation. While lectures and labs demand significant time and energy, they play a crucial role in helping students develop programming skills. However, these efforts may seem futile when weighed against the steep costs associated with failure and attrition. Many Indonesian university students appear to face similar challenges in their computer science courses, with a lack of familiarity with specific programming languages being a major obstacle to mastering fundamental programming concepts [6]. Students should also work on developing their skills in the three interconnected aspects of programming: grammar, design, and structure. Public university students in Indonesia face similar challenges when trying to learn computer programming. For those majoring in electronics engineering, programming is a required subject. However, for first-year students, mastering programming can be a daunting task. Learning programming languages is only the first step in this complex process. Additionally, learning programming requires university students to develop the ability to formulate a plan for solving a given problem [7]. Some fundamental concepts that students encounter in class include variables, arrays, and iteration. In their haste to earn a passing grade on an assignment, students often resort to poor programming practices.

Machine Learning (ML) technology has the potential to benefit various fields; however, its application in educational tools remains limited. This study integrates ML into a simulator designed to assist students in evaluating programming courses. ML can support decision-making, and in this system, it aids students in making informed choices while learning programming. The machine receives coding syntax as input from students, processes and analyzes it using pre-existing knowledge, and then provides feedback on the syntax. Based on this feedback, students can make decisions regarding their learning. The simulator employs the cosine similarity algorithm as its core technique. This algorithm assumes that features are conditionally independent given the class label. While this assumption may not always hold in practice, cosine similarity still produces reliable results and performs well on many real-world datasets.

This study investigates whether there is a difference in performance between high-anxiety and low-anxiety students when using a Machine-Learning (ML) simulator versus a non-ML programming simulator. It is hypothesized that students with high anxiety may benefit more from an ML-based simulator due to its adaptive and supportive features, which could enhance their learning experience. Conversely, low-anxiety students might perform similarly across both types of simulators, as they may require less external support. The study also explores whether the ML simulator mitigates the negative effects of anxiety, leading to improved performance compared to traditional simulators. If a significant difference is found, it could provide insights into designing more personalized educational tools for diverse learners. Ultimately, the findings may contribute to the optimization of programming education by addressing the impact of anxiety on student performance.

II. Literature Review

A. Performance in Programming

The term 'performance' refers to how well a person performs after completing a course of study or receiving therapeutic intervention. In computer science classes, common methods for evaluating students' progress include exams, quizzes, and homework. For example, evaluating students' progress highlighted the potential of using a simulator to support cognitive growth through self-experimentation, enhancing their understanding of the provided content. According to a study by Novak [8], students' preferences for reading the textbook, working in small groups with classmates, and completing course tasks showed a significant correlation with their post-test performance scores, suggesting that simulation-based learning could improve student performance. Additionally, students rated the simulation as more enjoyable than most other course components, including lectures, textbooks, homework, and group projects.

Research by Chernikova *et al*. [9] found that simulations using contemporary technologies had a significantly greater impact. There is no better approach to constructing learning environments in higher education than using simulations, which provide numerous opportunities for practice. Simulation-based learning can be integrated into academic programs at an early stage due to its effectiveness for students at all levels of study. According to a study by Isiaq and Jamil [10], engaging simulator-driven programming classes are linked to related careers and job opportunities. Students also described the classroom atmosphere as more welcoming and conducive to open discussion.

B. Anxiety in Programming

First-year computer science students often struggle to develop reliable mental models of how programs work. Another challenge they face is expanding their coding abilities beyond what they have learned so far. Many students

experience a growing aversion to programming after starting computer science courses, as these courses often alter their perception of the subject.

In the early days of computing, when computers were manually operated, programming languages were primarily used to organize simple operations. However, with the advent of object-oriented programming and procedural abstraction, computer intelligence advanced significantly. As a result, some students feel anxious about programming because, rather than modifying data through individual, straightforward actions, they must manage a series of processes as a unified whole. This anxiety negatively impacts academic performance and, in turn, affects student retention rates.

Uncontrolled anxiety negatively impacts students' academic performance, resulting in low achievement and high dropout rates [11]. Computing anxiety refers to the fear of performing poorly on computer-related tasks. It occurs when students have experienced or anticipate a loss of self-esteem in computing situations [12, 13]. Even as students gain experience, some still experience varying degrees of computer anxiety [13], which significantly hinders their ability to use computers efficiently. Speier *et al*. [14] found a correlation between high levels of anxiety before beginning computer learning and poor performance [14]. Students who fear computers tend to perform poorly academically and develop negative attitudes toward them.

An improper or erroneous perception of circumstances relieves anxiety. Students experience programming anxiety as a result of their incorrect belief in their own skills to master computer programming. This occurs due to the failure of the activation process [15], preventing students' mental schemas from providing the necessary foundation for solving programming challenges. Many freshmen struggle with computer programming because they may not have fully developed the cognitive models essential for the task. Several studies have examined the factors that influence a student's ability to learn computer programming.

However, some of these skills may not be fully developed when students begin their undergraduate programs, making them anxious about learning to code. In addition to feeling inadequate, students may experience increased anxiety when they receive negative feedback from computers, such as errors caused by improper program compilation. Anxieties about programming, a subset of computer anxiety, were assessed in this study by categorizing students' mean CPAQ (Computer Programming Anxiety Questionnaire) scores. However, to gain deeper insight into students' emotional states during programming activities, interviews or qualitative data could complement this approach. A low mean score suggests higher anxiety, while a high mean score indicates lower anxiety.

C. Constructivism in Teaching and Learning Programming

If students find learning programming challenging, their motivation, engagement, and classroom performance may decline. The majority of university students in this study struggled to create the correct syntax or code for several assigned challenges. They struggled to adapt to new problems because they had merely memorized the code provided by the professor. According to the study of Radosevic *et al*. [16], some students also wrote large amounts of software code without following proper syntax or conducting logical tests. This resulted in a high number of errors and discouraged them from continuing with programming Therefore, instructors need to completely revamp the way students learn. With the increasing use of technology and self-education in the modern world, applying constructivism in the classroom has become essential. Previous research indicates that constructivism enhances the learning environment, making it particularly valuable for programming courses [17–19]. Collaboration cognitive load theory offers a method to enhance cognitive processing by leveraging shared working memory, as explained by Looker [17], hen the complexity of a task exceeds an individual's working memory capacity.

Zhu [20] cites constructivist learning theory as the foundation for the idea that students actively and effectively learn by constructing mental models of their learning objectives. In other words, learning is an internal cognitive process in which students acquire new information by integrating it with their existing knowledge [21]. This learning philosophy centers on prioritizing the learner to achieve more personalized learning outcomes. In a constructivist classroom, students actively engage in expanding their understanding rather than passively absorbing information from their surroundings [22]. Instead of simply imparting factual information, teachers serve as guides while students learn. The content of textbooks becomes secondary to the goal of helping students build meaningful lives beyond school. Rather than merely assisting educators in delivering information, learning media foster the development of collaborative learning environments. One way students collaborate and learn through exploration is through conversation.

D. Persuasive Technology

The use of technological resources is more powerful than any persuasive ideology, as it offers new possibilities for alternative development in all its forms. Instead of viewing analytical technologies as mere tools, Pretto and Cláudio [23] stresses their significance. Each subject in education carries its own unique set of values, social practices, conventions, and traditions, which shape human-to-human interactions [24]. Every person's actions stem from their unique fears and aspirations or their response to predetermined goals [25]. As a major component of assessing motivation and competence, this study's approach examines behavior. The reason is that people learn to react in particular ways based on their culture. Consequently, certain actions can be anticipated and influenced accordingly. According to Cialdini [26], persuasive communication is a method of persuasion that involves communicating with the goal of getting someone to adopt a certain behavior, concept, or belief system by making use of logical-rational or symbolic resources.

Both the world and technology have undergone significant changes since Carl Hovland began researching persuasive communication in the 1950s [27]. Technology has revolutionized communication, and computers have become indispensable in modern life. However, the art of persuasion remains unchanged. Today, persuasive computing systems

are used to influence people's thoughts and behaviors. Persuasive technology refers to systems that rely on social influence and persuasion—rather than force or coercion—to shape opinions and actions Persuasive technology's impacts are intended rather than unanticipated.

The term 'persuasive technology' refers to any computer program or information system designed to encourage a change in attitude or behavior without relying on deception or coercion.

A persuasive system consists of three main components [27]. The first step in developing such a system is understanding the fundamental issues that provide its context. The second component involves examining the context of the persuasive system itself. This includes not only the event and strategy used but also the system's overall goal. Intention plays a key role, encompassing both the type of change being pursued and the persuader's role in influencing it. Every aspect of the event is shaped by the user's context, the technology's context, and its application. It is essential to distinguish between the strategic environment, the message, and its method of dissemination.

The final stage involves evaluating system attributes, both for newly developed persuasive technologies and for assessing features in existing systems. These system attributes ultimately determine the effectiveness of the persuasive technology. Evaluation of an existing system's qualities is the intended use of system attributes. According to Oinas-Kukkonen [28], this level is all about helping with main tasks and conversations, making the system more credible, and having social support.

Persuasive technology is not a new concept in education. Its use in this field dates back quite some time. Recognizing its potential in education, BJ Fogg became a pioneer in persuasive technology. To teach students about the dangers of unsafe sexual behavior, Fogg [27] employs the games HIV Roulette and Baby Think It Over, the latter of which encourages young women to delay having children. Both approaches aim to educate and inspire university students, ultimately influencing their actions. In this study, persuasive techniques are employed to enhance students' motivation, engagement, and performance. Educational tools such as Google Classroom, Edmodo, and iTunes U serve as valuable learning aids.

Computers have become an integral part of human life, and technological advancements significantly impact communication. As a result, persuasion is also deeply embedded in modern technology. We are currently living in an era of persuasive technology—interactive computer systems designed to influence people's thoughts and behaviors. Persuasive technology refers to systems that alter user attitudes or behaviors through social influence and persuasion rather than force. Its outcomes are not unintended byproducts but rather its primary objectives.

Persuasive technology can be categorized into three main types. To classify how individuals experience and interact with computer technology, Fogg [27] introduces the Functional Triad. Computers can function in three primary roles: as tools, as social actors, and as mediums. These roles can operate individually or in combination, depending on the context. Technology facilitates reorganization processes, enhancing people's ability to engage in desired activities. Interactive and narrative-driven media can create immersive experiences that promote behavior practice, develop empathy, or establish causal relationships. Additionally, technology can act as a social agent, influencing the behavior of those it is designed to support. It achieves this through social cues such as language, the assumption of social roles, and physical presence.

The primary tool developed for this research is the ML-programming simulator, designed to assist students with their programming coursework. Rather than relying on coercion, the ML-programming simulator leverages social influence and persuasion to shape users' habits and perspectives. It is expected that using the simulator will enhance students' learning outcomes.

Additionally, the ML-programming simulator was built upon established theories and principles known to boost student engagement and motivation. As a result, its use is intended to foster these qualities. Students actively engage with the simulator throughout three phases: pre-class, in-class, and post-class. The researcher carefully selected the learning components in collaboration with an academic expert in the field. The simulator is designed to inspire students through its rich collection of educational resources. The overarching goal of its development was to enhance student engagement, performance, and motivation to learn.

Persuasive technology in education consists of two main branches: the theoretical branch, which explores the literature on the subject, and the practical branch, which applies the technology as a tool to support both instructors and students.

III. Methodology

This study's research population was selected using a non-probability sampling method designed to be representative. The participants were undergraduate students in Indonesian computer science programs who were required to take programming classes. Because both the students enrolled in these classes and the participating lecturers were readily available and under the researchers' control, a representative sample was used. To validate and analyze the data collection process, the study involved three lecturers with over a decade of experience teaching programming.

Sampling refers to the process of selecting a subset of a population to obtain a statistically valid cross-section [29]. Since the target population is typically too large to include in its entirety, sampling is a crucial technique in scientific research. A well-chosen sample provides a sufficiently large subset of the target population to effectively answer the study's research question.

For this study, the people of Medan are considered representative of all Indonesians. Universitas Negeri Medan was selected through careful selection as the university with the attribute of having programming as a mandatory course.

In this study, universities were selected using a purposeful sampling method, while undergraduate participants were chosen through random sampling. The study participants were limited to university freshmen. Each class in the experiment consisted of students from the preceding two years, 2021 and 2022, resulting in a total sample of approximately 120 university students. All participants were computer science undergraduates selected at random.

One objective of descriptive analysis is to objectively

characterize the type and intensity of sensory qualities. This approach was revolutionary as it provided a scientific foundation for sensory evaluation by generating objective, statistically sound, and analyzable data. It remains an essential method in modern sensory analysis and has led to the development of various descriptive analytic techniques. Traditional descriptive methods, such as profiling-based approaches and quantitative descriptive analysis, involve trained expert panels who subjectively assess the quality and intensity of sensory properties in samples. In recent years, there has been a rise in faster descriptive techniques, including polarized sensory placement, projective mapping, and overall similarity/variation analysis, which rely on inexperienced consumers to evaluate sensory attributes. Due to its effectiveness and adaptability, descriptive analysis continues to be widely used. Descriptive analysis provides sensory data that is thorough, accurate, reliable, and objective. It relies on human assessors in highly controlled settings to minimize bias. Assessors with exceptional sensory abilities are carefully selected and undergo a rigorous six-month training program in traditional methodologies, including profiling-based approaches and Quantitative Descriptive Analysis (QDA), to ensure the generation of validated data. According to Alessi and Stanley [30], individuals who complete this training can consistently rate perceived intensity and quality, both independently and in comparison with other evaluators. Advanced methods such as Free Choice Profiling (FCP), flash profiling, sorting, projective mapping, and Polarized Sensory Positioning (PSP) may involve untrained consumers who lack prior knowledge or experience. These methods categorize products based on broad similarities or differences. Measurements can be conducted after product identification and labeling, or products can be categorized before being assigned group names [31].

Descriptive analysis is a valuable tool in quasi-experimental research, where it is used to meticulously describe a scenario or topic of interest. Quasi-experimental research aims to establish a cause-and-effect relationship between two variables (independent and dependent) without employing random assignment. Instead, subjects are categorized based on predefined criteria. This approach is particularly useful when true experimental conditions are not feasible due to constraints such as limited funding, scheduling difficulties, or the absence of randomized controlled trials. To better understand the characteristics and current state of relevant variables in quasi-experimental research, descriptive analysis is applied within the study's design framework. Common descriptive research methods used in quasi-experimental studies include population surveys, public opinion polls, status studies, surveys and interviews, observational studies, job descriptions, literature reviews, documentary analyses, anecdotal records, critical incident reports, test score analyses, and normative data collection.

When random assignment of participants is not possible, researchers attempt to replicate true experimental conditions as closely as possible. Within these constraints, they must acknowledge and account for any compromises to the study's internal validity. Several quasi-experimental designs have gained prominence, including pre-posttest, interrupted time-series, and nonequivalent groups. Finally, quasi-experimental research relies heavily on descriptive analysis, which provides a detailed understanding of the situation or issue under investigation. It helps researchers assess the characteristics and status of variables within the study's framework, informing future research and interventions while offering critical insights into the relationships between variables.

A. Research Instruments

This study utilizes an instrument to measure students' performance in a programming class based on the questions set by instructors for the final exam. The researcher administered the same assessment at both stages to evaluate learning progress. The test consists of five questions derived from the material covered in a one-hour programming class, specifically focusing on looping. Identical questions are used in both the pre-test and post-test [32]. Computer programming lecturers verified the accuracy of the instrument's content, ensuring that the questions effectively assessed students' understanding of programming concepts—particularly looping—before approving them.

When students are unable to appropriately analyze a situation, our anxiety levels remain elevated. Due to an inaccurate assessment of their ability to learn computer programming, university students often experience programming anxiety. When the activation process fails to occur, students are unable to develop the necessary mental schemas to analyze programming challenges and formulate solutions [15]. While most freshmen have a basic understanding of computer programming fundamentals, they may lack the fully developed cognitive models required for advanced programming. This study investigates the variables that influence a learner's proficiency in computer programming.

Additionally, students may feel unprepared for the challenge of learning programming if they have not fully developed specific skills before entering their undergraduate program. Frequent negative feedback from computers—such as repeated compilation errors—can cause students to doubt their competence in certain areas. In this study, programming anxiety, a subset of computer anxiety, refers specifically to situational anxiety experienced in programming-related tasks.

B. Design and Development

The application is currently in the design phase of development, where concepts are transformed into a preliminary sketch. According to Ashmore *et al*. [29], the design process includes the following steps: (i) brainstorming potential topics, (ii) analyzing tasks and concepts, and (iii) outlining the program.

1) Initial content ideas

Generating content ideas early is essential for establishing foundational concepts related to the appropriate material and learning techniques used in application development. This section outlines the strategic approaches for delivering content within the application, known as Persuasive Design Principles. Additionally, the researcher explores the implementation of Jonassen's Constructivist Learning Environments (CLEs) in greater detail [33]. The

programming topic that was selected in this study is Looping.

2) The elements of effective design

Persuasive Technology is the study of attitudes and the methods used to influence them [26]. The macro approach to multimedia design in this study is based on three key principles: similarity, suggestion, and tailoring.

3) The similarity principle

According to Cialdini [26], university students are more influenced by AI systems that share certain characteristics with them. Based on this insight, the researcher developed a program functionally equivalent to the tools students commonly use in class. In programming courses, students frequently work with text editors such as Sublime Text or Notepad++. To maintain familiarity, the researcher designed the software to allow students to write syntax in the same way they do during class, eliminating the need for additional time to learn a new interface. Unlike traditional class tools, where users must first enable XAMPP to execute code and check for syntax errors, the researcher developed a simulator software that allows students to run their syntax and receive immediate feedback. This enhancement streamlines the coding process and improves the learning experience as shown in Fig. 1.

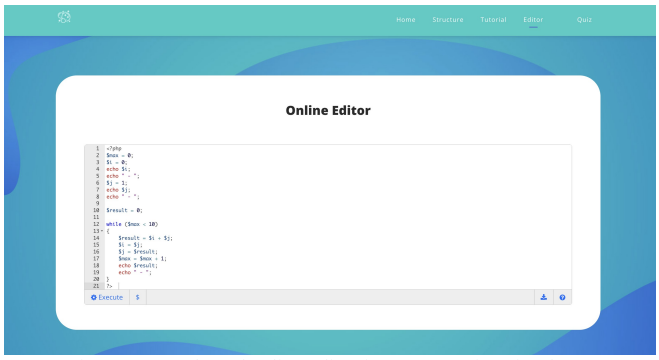

Fig. 1. Screenshot of online editor in programming simulator.

4) The basics of proposal

According to Cialdini [26], a computer program's persuasiveness can be enhanced by strategically timing its suggestions. The Suggestion Principle was applied in the development of this programming simulator to support students in learning programming. When the simulator detects syntax errors, it provides tailored suggestions based on students' previous mistakes. As illustrated in Fig. 2, the simulator also offers advice and tips for correcting syntax errors.

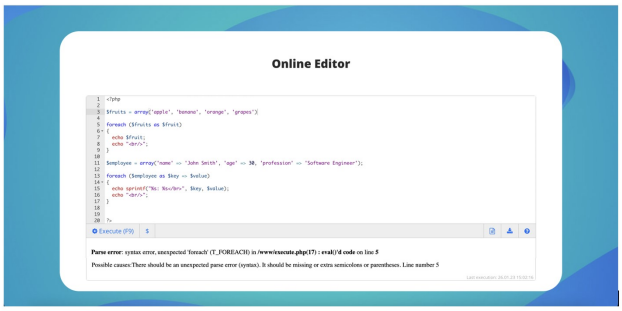

Fig. 2. Screenshot of the suggestion from programming simulator.

5) Tailoring principle

The Tailoring Concept suggests that an application's persuasiveness increases when its content is designed and adapted to meet the user's needs, interests, personality traits, and context. Following Loranger's guidelines, the researcher developed a programming simulator based on these principles. These guidelines are particularly useful for creating applications that effectively support students' learning. One key strategy involves using familiar terms and concepts that align with the students' prior knowledge. Additionally, students appreciate the software's simple and user-friendly interface as shown in Fig. 3.

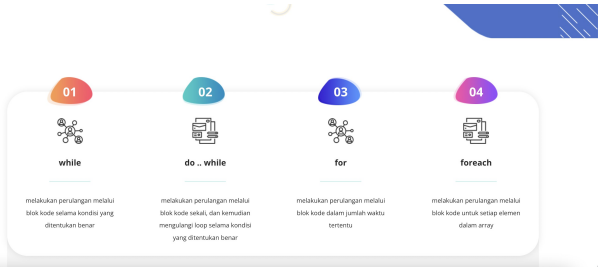

Fig. 3. Screenshot of programming simulator using Indonesian languages.

6) Learning environment based on constructivism

In 1999, Jonassen introduced the concept of Constructivist Learning Environments (CLEs), which emphasize problem-based learning. This approach encourages students to solve problems by drawing from various resources, including the problem itself, similar cases, informational materials, cognitive tools, discussion and collaboration platforms, and social contextual support.

To ensure that the programming simulator fosters problem-solving skills and cognitive development, this study modified four out of the six CLE components. These components are outlined below.

7) Problem

Students often struggle with looping, a fundamental concept in computer programming. According to the researcher's previous problem analysis, looping is one of the most challenging concepts for students to master. Thus, the researcher in this study focuses on looping as a topic to assist students' comprehension using this simulator.

This programming simulator is ideal for teaching trainees loop syntax because the simulator provides immediate feedback on whether the code is correct. If the syntax is valid, it displays the code's output. If an error is detected, the simulator highlights the issue and specifies its location.

8) Cases that are relevant

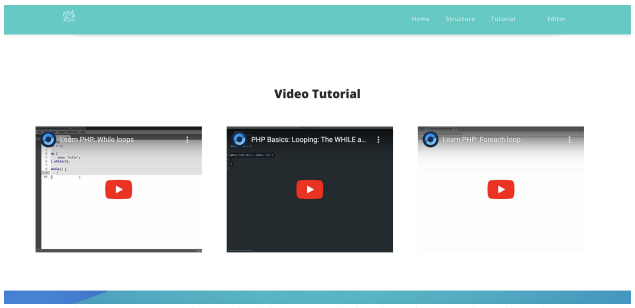

Fig. 4. Screenshot of video tutorial of coding using looping.

This program highlights similar scenarios through video tutorials and an online editor. To facilitate case-based reasoning and enhance cognitive flexibility, learners should

be exposed to relevant examples or worked solutions [32]. In this programming simulator, related scenarios demonstrate how loops are implemented in code.

Providing examples that closely align with students' own challenges, contexts, and expected outcomes strengthens their memory scaffolds. To find the correct solution, students are encouraged to compare the highlighted examples to the issues they are currently facing. As illustrated in Fig. 4, the programming simulator includes video tutorials on looping as related case examples.

9) Data storage services

Students can also access a wealth of informational resources through this app. According to Witzel *et al*. [32], designers should provide only the most relevant information to help students solve problems and refine their understanding. Based on their proficiency level, students can prioritize syntax related to the looping topic as shown in Fig. 5. Additionally, they can revisit previously provided syntax as needed for reinforcement.

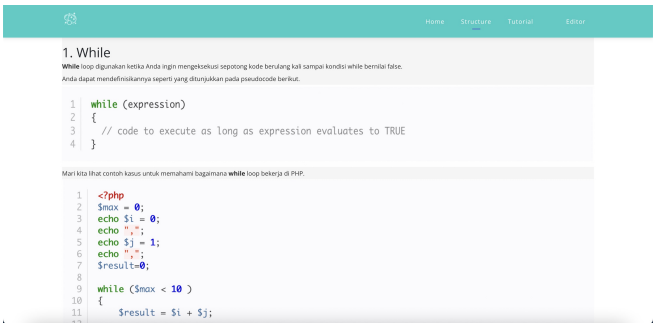

Fig. 5. Screenshot of theories of while—looping.

10) Cognitive resources

Key components that support the development of essential skills include cognitive tools such as information-gathering tools, problem-modeling tools, performance support tools, and knowledge modeling techniques [32]. According to Kommers *et al.*, cognitive tools are generalizable computer programs designed to facilitate and engage cognitive processes.

In this study, the programming simulator functions as a problem-representation tool, helping learners construct mental models of how code behaves and interacts. Additionally, it serves as a knowledge modeling tool, enabling students to demonstrate their understanding of programming concepts.

C. Task and Concept Analysis

After establishing the preliminary concept for the program content, task and concept analyses were conducted in this study. Task analysis involves identifying the data flow, inputs, and decision-making processes required to perform a specific action. Access to relevant information enables the researcher to gain a comprehensive and precise understanding of the problem.

A programming expert contributed to the concept analysis in this study. Based on this analysis, the researcher identified a key programming challenge to address in the simulator: enhancing students' performance, engagement, and perceived motivation in programming courses, particularly in the topic of looping.

D. Program Description

This study utilized a custom-built programming simulator to enhance students' knowledge and perceived motivation, thereby increasing engagement in a programming course. The target population consists of university students from the 2020 and 2021 cohorts, aged 17 to 20 years.

The programming simulator was designed to improve students' performance, engagement, and perceived motivation in programming. It comprises four main components, detailed as follows:

1) What is Looping? Provides a definition of looping, explains its execution process, and presents examples.
2) The Framework of Iteration: Educates students on the structure and implementation of loops in programming.
3) Instruction: Includes video tutorials demonstrating how to code loops.
4) Task: Contains simulator-based reinforcement exercises designed to strengthen students' understanding and motivation.

IV. Results and Discussion

With this study, the researcher examined how the dependent variable was influenced by two independent factors: the ML programming simulator and the noML (non-Machine-Learning) programming simulator. To determine whether the presentation method significantly affected student performance, a one-way Analysis of Variance (ANOVA) was conducted. Each group's pre- and post-test results were averaged, and their variability was assessed before computing the computer's performance. As shown in Table 1 of the descriptive statistics, 60 students used an ML programming simulator, while 60 students were in a control group that did not use an ML programming simulator. After using the ML programming simulator, the average score increased from 74.20 on the pretest to 87.45 on the posttest. The average pre- and post-test scores for all ML programming simulator users were 74.34 and 84.98, respectively. When comparing the ML programming simulator with the noML programming simulator, the average difference between the two sets of scores was 10.64 points, whereas the ML programming simulator group exhibited a mean difference of 13.20 points. These results suggest that students who used the ML programming simulator performed better than those who used the noML programming simulator. Fig. 6 presents a bar graph contrasting the performance of each mode (NoML, ML, and Total) on the pretest and posttest. The standard deviations are shown by error bars.

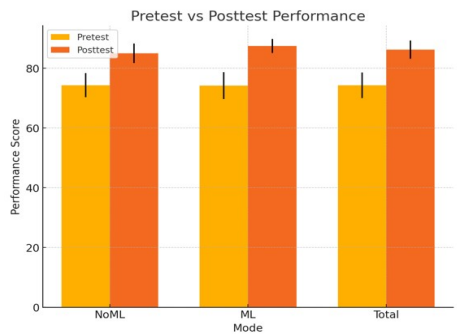

Fig. 6. Graph of students performance using ML and noML programming simulator.

Table 1. Results of students' performance of NoML programming simulator and ML-programming simulator

Mode	Unit	Pretest Perf	Posttest Perf
NoML Programming Simulator	N	60	60
	Mean	74.34	84.98
	Std. Deviation	4.06	3.28
	Std. Error of Mean	0.524	0.424
ML Programming Simulator	N	60	60
	Mean	74.20	87.45
	Std. Deviation	4.49	2.33
	Std. Error of Mean	0.579	0.300
Total	N	120	120
	Mean	74.27	86.22
	Std. Deviation	4.26	3.09
	Std. Error of Mean	0.389	0.282

N = sample size
Std. Deviation = Standard Deviation
Std. Error of Mean = Standart Error of Mean

However, while the findings highlight a notable improvement in student performance with the ML programming simulator, it remains unclear whether this improvement is due to the simulator itself or potential artifacts arising from its implementation. Factors such as differences in instructional design, unintended biases in simulator behavior, or variations in user interaction with the simulators may have influenced the results. Future studies should carefully examine these variables to ensure that the observed differences are genuinely attributable to the ML programming simulator rather than extraneous factors inherent to the experimental setup.

As a result, the hypothesis of this study, which stated that there is no difference in student performance when using an ML programming simulator versus a noML programming simulator, is rejected. This suggests that university students instructed using an ML programming simulator outperformed those who were not. The ANOVA test results in Table 2, indicate a statistically significant difference between the two groups, with an F-value of 22.518 and a *p*-value of 0.001, which is well below the 0.05 threshold. These results confirm that students using the ML programming simulator performed significantly better than those using the NoML programming simulator. The independent samples T-test further supports this conclusion as shown in Table 3, with a Sig. (2-tailed) value of 0.001, indicating that the difference between the two groups is not due to chance. However, given the potential for implementation-related artifacts, additional research is needed to rule out alternative explanations for the observed performance improvements.

In addition, the output table above reveals a "Mean Difference" value of −2.468. This value, which falls within the 95% confidence interval of the difference between the two groups, represents the difference in average student learning outcomes between the ML programming simulator group and the noML programming simulator group.

Table 2. Anova test of performance score

Source	Type III Sum of Squares	Df	Mean Square	F	Sig.
Corrected Model	182.732[a]	1	182.732	22.518	<0.001
Intercept	892104.617	1	892104.617	109932.484	<0.001
GROUP	182.732	1	182.732	22.518	<0.001
Error	957.573	118	8.115		
Total	893244.921	120			
Corrected Total	1140.305	119			

df = Degrees of Freedom; F = F-Value; Sig. = Significance Value; a = R Squared = 0.219 (Adjusted R Squared = 0.214)

Table 3. t-test of performance score

	t-test for Equality of Means						
	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
						Lower	Upper
Posttest Perf	−4.745	118	<0.001	−2.468	0.520	−3.498	−1.438

t = t-value; df = Degrees of Freedom; Sig. = Significance Value

These results have important implications for programming education. While traditional programming simulators are effective, ML-based simulators may offer advantages in interactivity and adaptability. However, to ensure that the observed improvements are truly due to the ML enhancements rather than implementation artifacts, future research should explore additional factors influencing student success, such as instructional methods, user interactions, and software biases. Moreover, incorporating qualitative measures, such as student feedback and behavioral observations, could provide deeper insights into how ML-driven simulators affect student motivation and problem-solving skills.

V. Conclusion

The study results indicate that students using a ML Programming Simulator outperformed those who did not, supporting the first hypothesis. Additionally, a significant difference was observed in student performance between the control group without a ML programming simulator and the treatment group with a ML programming simulator. Students in the treatment group demonstrated significantly higher levels of performance with the material compared to those in the control group.

These findings highlight the role of machine learning in fostering active participation in programming classes. As a result, two distinct types of programming simulators emerged in this study: the ML programming simulator and the noML programming simulator. Performance test results further confirmed that students who used the ML programming simulator achieved significantly higher scores than those who did not.

The study underscores the transformative impact of ML programming simulator on student performance and learning outcomes in programming classes. Students with greater exposure to ML programming simulator exhibited higher leverls of performance than their peers using traditional simulator.

Simulator, in genereal, replicate real-world conditions and processes, serving as step-by-step guides through various system occurences. Different simulation programs and techniques exist across various fields, each designed to

enhance learning experiences. Despite challenges in understanding assembly and programming, simulations have proven effective in creating meaningful and engaging learning environments across multiple educational domains.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Author Wan Ahmad Jaafar Wan Yahaya conceptualized the study, designed the methodology, and supervised the overall project. Author Tansa Trisna Astono Putri conducted the data collection, performed the initial analysis and contributed to drafting the manuscript. Author Nur Azlina Mohamed Mokmin assisted with data interpretation, statistical analysis and provided critical revisions to the content. Author Sriadhi Sriadhi managed the literature review and participated in the final editing and formatting of the manuscript. All authors read and approved the final version of the article.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the Centre for Instructional Technology and Multimedia, Universiti Sains Malaysia, and the Information Technology and Computer Education Study Program, Universitas Negeri Medan, for their valuable support and collaboration. Their contributions and facilitation were instrumental in enabling this research to be successfully conducted.

REFERENCES

[1] Bureau of Labor Statistics. (September 2022). Employment projections: 2021–2031 summary. *Bureau of Labor Statistics.* [Online]. Available: https://www.bls.gov/news.release/ecopro.nr0.htm

[2] M. Primayunita. (March 2023). Indonesian programmer salary 2023: Promising career opportunities. *Dicoding*. [Online]. Available: https://www.dicoding.com/blog/gaji-programmer-indonesia-2023-peluang-karier-menjanjikan/ (in Indonesian)

[3] C. Angeli and N. Valanides, "Computers in human behavior developing young children's computational thinking with educational robotics: An interaction effect between gender and scaffolding strategy," *Computers in Human Behavior*, vol. 105, 105954, 2020.

[4] U. Nikula, O. Gotel, and J. Kasurinen, "A motivation guided holistic rehabilitation of the first programming course," *ACM Transactions on Computing Education*, vol. 11, no. 4, pp. 1–38, Nov. 2011.

[5] M. Corney, D. Teague, and R. Thomas, "Engaging students in programming," in *Proc. Twelfth Australasian Computing Education Conf.*, 2010, pp. 63–72

[6] A. Baist and A. S. Pamungkas, "Analysis of student difficulties in computer programming," *VOLT: Jurnal Ilmiah Pendidikan Teknik Elektro*, vol. 2, no. 2, pp. 81–92, Oct. 2017.

[7] I. N. Umar and T. H. Hui, "Learning style, metaphor and pair programming: Do they influence performance?" *Procedia-Social and Behavioral Sciences*, vol. 46, pp. 5603–5609, 2012.

[8] E. Novak, "Effects of simulation-based learning on students' statistical factual, conceptual and application knowledge," *Journal of Computer Assisted Learning*, vol. 30, no. 2, pp. 148–158, July 2013.

[9] O. Chernikova *et al.*, "Simulation-based learning in higher education: A meta-analysis," *Review of Educational Research*, vol. 90, no. 4, pp. 499–541, June 2020.

[10] O. Isiaq and M. G. Jamil. (2017). Exploring student engagement in programming sessions using a simulator. *ICICTE*. [Online]. pp. 206–215. Available: http://www.icicte.org/ICICTE_2017_Proceedings/6.3_Isiaq%20_%20Jamil.pdf

[11] M. J. Acelejado. (2003). The impact of using technology on students' achievement, attitude and anxiety in mathematics. [Online]. Available: https://acasestudy.com/the-impact-of-using-technology-on-students-achievement-attitude-and-anxiety-in-mathematics/

[12] V. McInerney, "Computer anxiety: Assessment and treatment," Ph.D. dissertation, Western Sydney University, New South Wales, 1997.

[13] G. A. Marcoulides, "The relationship between computer anxiety and computer achievement," *Journal of Educational Computing Research*, vol. 4, no. 2, pp. 151–158, May 1988.

[14] C. Speier, M. G. Morris, and C. M. Briggs, "Attitudes toward computers: The impact on performance," in *Proc. AMCIS 1995*, 1995, p. 43.

[15] R. E. Mayer, "The psychology of how novices learn computer programming," *ACM Computing Surveys (CSUR)*, vol. 13, no. 1, pp. 121–141, Mar. 1981.

[16] D. Radosevic, T. Orehovacki, and A. Lovrencic, "Verificator: Educational tool for learning programming," *Informatics in Education*, vol. 8, no. 2, pp. 261–280, Oct. 2009.

[17] N. Looker, "A pedagogical framework for teaching computer programming: A social constructivist and cognitive load theory approach," in *Proc. 17th ACM Conf. on International Computing Edu Cation Research*, 2021, pp. 415–416.

[18] F. J. Agbo *et al.*, "Examining theoretical and pedagogical foundations of computational thinking in the context of higher education," in *Proc. 2021 IEEE Frontiers in Education Conf. (FIE)*, 2021, pp. 1–8.

[19] C. S. González-González *et al*., "COEDU-IN project: An inclusive co-educational project for teaching computational thinking and digital skills at early ages," in *Proc. 2021 International Symposium on Computers in Education (SIIE)*, 2021, pp. 1–4.

[20] C. Zhu, "E-learning, constructivism and knowledge building," *Educational Technology*, vol. 48, no. 6, pp. 29–31, 2008.

[21] S. O. Bada, "Constructivism learning theory: A paradigm for teaching and learning," *Journal of Research and Method in Education*, vol. 5, no. 6, pp. 66–70, Dec. 2015.

[22] Z. Xu and Y. Shi, "Application of constructivist theory in flipped classroom—Take university english teaching as a case study," *Theory and Practice in Language Studies*, vol. 8, no. 7, pp. 880–887, July 2018.

[23] N. Pretto and C. P. D. Cláudio, "Technologies and new educations," *Brazilian Journal of Education*, vol. 11, pp. 19–30, 2006.

[24] R. D. Santos, "Three fundamental relationships in higher education," *Revista Iberoamericana de Educacion*, vol. 36, no. 9, pp.1–9, Sep. 2005. (in Portuguese)

[25] K. Hogan, *The Psychology of Persuasion: How to Persuade Others to Your Way of Thinking*, Louisiana: Pelican Publishing, 1996, ch. 1.

[26] R. B. Cialdini, *The Weapons of Persuasion: How to Influence and Not Be Influenced*, Rio de Janeiro: Editora Sextante, 2012, ch. 1.

[27] B. J. Fogg, "Computers as persuasive social actors," *Persuasive Technology; Using Computers to Change What We Think and Do*, Massachusetts: Morgan Kaufmann, 2002, ch. 1, pp. 89–120.

[28] H. Oinas-Kukkonen, "Requirements for measuring the success of persuasive technology applications," in *Proc. 7th International Conf. on Methods and Techniques in Behavioral Research*, 2010, pp. 1–4.

[29] R. Ashmore, R. Calinescu, and C. Paterson, "Assuring the machine learning lifecycle: Desiderata, methods, and challenges," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–39, May 2021.

[30] S. M. Alessi and R. T. Stanley, *Multimedia for Learning: Methods and Development*, Boston: Allyn & Bacon, Inc., 2000, ch. 1.

[31] M. G. Jamil and S. O. Isiaq, "Teaching technology with technology: Approaches to bridging learning and teaching gaps in simulation-based programming education," *International Journal of Educational Technology in Higher Education*, vol. 16, no. 1, p. 25, Aug. 2019.

[32] B. S. Witzel, C. D. Mercer, and M. D. Miller, "Teaching algebra to students with learning difficulties: An investigation of an explicit instruction model," *Learning Disabilities Research and Practice*, vol. 18, no. 2, pp. 121–131, April 2003.

[33] D. H. Jonassen and L. R. Murphy, "Activity theory as a framework for designing constructivist learning environments," *Educational Technology Research and Development*, vol. 47, pp. 61–79, Mar. 1999.

Copyright © 2025 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).