[1]Kapil Shukla
[2]Parag Shukla

An Automatic Evaluation of the Programming Solution Submitted by the Learner for Suggesting the Appropriate Next Programming Task

Abstract: - Online coding platforms have been available for learning programming topics or completing courses. Learner needs to submit solution of provided task after checking all test cases. Some Institutes and programming teachers are either hosting their own open-source coding platform or using an online coding platform for teaching programming subjects. It is required to have the information in the form of automatic feedback obtained from their submission's evaluation about how the learners are performing in programming topics. Our work introduces a model that aims to automate the process of suggesting a sequence of coding tasks for the learners to complete programming courses according to the knowledge and skill possessed by the learner. Additionally, this model provides a means for evaluating the performance of each learner in different topics covered by the course instructor. By utilizing this model, course instructors can obtain valuable insights into each learner's performance in specific course topics.

Keywords: online programming course, learners' performance, automatic code evaluation

I. INTRODUCTION

Online programming courses have gained immense popularity as a prevalent method for introducing learners to the programming world [1]. With the increasing accessibility of online programming classes, thousands more students are drawn to them each year. In these courses, learners address programming challenges by providing incomplete solutions and verifying the results [2]. Predicting the performance of students' problem-solving and learning skills is indeed feasible [3]. Recent advances in information and communication technologies have changed the methods by which learners gain knowledge. There are a multitude of advantages associated with e-learning, some of which include the availability of course materials online and the removal of the need for students to physically attend classes in a conventional classroom environment. E-learning has gained a lot of popularity and is essential to the expansion of web-based education systems, in addition to being less expensive than conventional methods of instruction. [4].

This is easy to do with automatic marking systems because they compare students' work to a set of prepared test cases. The test cases are assigned weights that reflect their level of difficulty [5]. A detailed system created to gather information on students' programming activities provides essential insights for educators. By monitoring metrics like coding frequency, completion rates, and problem-solving strategies, this system provides educators with a detailed insight into each student's progress as well as overarching class trends. Furthermore, by utilizing detailed reports, educators acquire practical insights to modify their teaching approaches, recognize students in need of support, and provide prompt assistance. With this extensive collection of data, educators are equipped to create a more supportive learning atmosphere, adeptly steering the learner towards achieving proficiency in programming concepts and skills [6]. Investigating the performance of novice learners in a basic programming language can yield insights into the dropout rate from the course, providing essential information for educators [7]. Spotting students at risk early in the course term benefits both instructors and the learner. Instructors gain insight into which students require extra support, while novices have a chance to receive support promptly, possibly enhancing their course outcomes [8].

Any code evaluation system must understand the type and implication of errors within the code. Providing summative feedback throughout programming can benefit in minimizing both syntax and semantic errors [9]. The core of automated code evaluation pivots on the capability to measure assessment targets in numerical terms. In this context, the system must be able to assign numerical values to various features of the assessment criteria,

[1] Research Scholar, Atmiya University, Rajkot, India. Email: 15616221002@atmiyauni.edu.in
[2] Associate Professor, School of Computing, Kaushalya – The Skill University, Ahmedabad, India. Email: paragshukla007@gmail.com

enabling objective evaluation of student performance. This numerical quantifiability simplifies the process of analyzing, comparing, and providing feedback on students' work proficiently and excellently [10].

To facilitate the recommendation of suitable programming tasks for development in automatic systems, it is crucial to develop models that can assess performance based on predefined criteria. These models can leverage the extensive datasets generated by the learners when they complete programming tasks [11, 12]. Learner is allowed to submit solution multiple times before successfully solving a task, which is depended on problem complexity level, are counted for each exercise performance evaluation.

Through its ability to facilitate the continuous monitoring of student performance, the model makes it possible for the computation of academic progress to change in a dynamic manner throughout the course of time. Multiple tests, which often give mixed results, are not needed with this flexible method because it takes into account the different ways a student learns. Teachers can get a good idea of how their students are growing by looking at data and getting feedback. This makes the learning environment more open and flexible [13].

We have designed a mathematical model that determines how well the student is performing on each computing problem and suggests the next one he or she should attempt to solve. Several parameters are considered by this model to determine the next best job for the student.

II. METHODOLOGY

A. Dataset

We utilized a system that exploits multiple clusters of programming tasks categorized by some topics, such as loops and conditions. Each cluster contains programming tasks with varying levels of complexity. We kept records of 30 learners' submissions for the programming tasks; the records included the number and type of errors made, successful or unsuccessful attempts, and the current complexity level of the task.

Based on that, we created an algorithm which can suggest the next statement of the problem to be given to the learner. We consider both the number and type of errors committed and failures on previous attempts and the level of complexity currently at this stage. Thus, recommendations can be provided for programming tasks best suited to allow learners to progress further in the course.

B. Proposed Model

We utilized a system that exploits multiple clusters of programming tasks categorized by some topics, such as loops and conditions. Each cluster contains programming tasks with varying levels of complexity. We kept records of 30 learners' submissions for the programming tasks; the records included the number and type of errors made, successful or unsuccessful attempts, and the current complexity level of the task.

Our research targets the development of a personalized model for programming task recommendations: one that takes into consideration the learner's current skill and learning goals. This shall be able to suggest relevant programming tasks at an appropriately challenging level to help the learner improve command over the subject.

By using this approach, we give learners the capability to start their course from a topic-based activity at any level of complexity, and then the model suggests which ones align with the chosen topic of interest by them so that the suggested activity fulfills the needed learning in such aspects related to the learner's profile level of proficiency. Of course, this is for helping learners along their road of improving programming skills toward realization of intended learning goals.

Our research identifies several relevant factors that influence the selection of the programming problem to be done next by the students.

1) The count of failed trails

2) The quantity of error detected in the problem solution

The learner is provided with a predefined number of trials for each task, which increases as the complexity level of the code progresses. The model suggests a new problem to the learner in two different scenarios. First, when the learner successfully executes the code, indicating a sufficient level of knowledge. Second, when the remaining number of trials for a particular problem reaches zero, encourages the learner to move on to a new challenge.

By incorporating these factors and implementing a dynamic recommendation system, we aim to provide learners with a well-rounded and advanced learning experience in programming.

1) The count of failed trails:

The model considers the number of unsuccessful attempts made by the learner while solving a particular problem. This helps gauge the learner's progress and determine if they need to continue with the same problem or move on to a new one.

To determine the correctness of a solution S, we rely on the outcome of its test cases. A solution is considered correct only if all the test cases pass. To represent this, we define a correctness score, denoted as $f_c(S)$. If all the test cases pass, the maximum correctness score can be 1, indicating a correct solution. However, if even a single test case fails, the correctness score is set to 0, indicating an incorrect solution. By looking at the complexity of the task, some numbers of trials are provided to the learner, t_{max}.

Every task is set with a maximum number of trials that the learner can take, t_{max}. Leaner can try to code the perfect solution for t_{max} times. As the value of actual trial t_{act} will increase, it will decrease the correctness score $f_c(S)$ from 1 to 0.

Calculation of the correctness score, $f_c(S)$ will be calculated as follows.

$$f_c(S) = t_{max} /(t_{max} + t_{act}), \text{ if } t_r >= 0,\ t_r = t_{max} - t_{act} \qquad \ldots(1)$$

2) The quantity of error detected in the solution:

The model analyzes the types of errors encountered in the learner's code. By understanding the specific areas where the learner is struggling, the model can suggest programming problems that target those areas for improvement.

To calculate the error score of a program S, we calculate it using the formula

$$f_e(S) = e_{max} / (e_S + e_{max}) \qquad \ldots(2)$$

where e_S represents the number of compilation errors encountered for program S, and e_{max} is the maximum number of compilation errors under consideration [14].

The error score is a measure of how well a program performs in terms of compilation errors. It is derived by dividing the maximum numbers, e_{max}, of allowed errors by sum of maximum number of error and actual error, e_S, come across while reaching to solution of provided task. This normalization ensures that the error score falls within the range of 0 to 1, with higher scores indicating fewer compilation errors and better performance.

By using this error score metric, we can assess the quality and reliability of a program based on the number of compilation errors it exhibits, relative to the maximum number of errors observed among all programs [14].

C. Final Performance Calculation

The final performance of learner for specific task from successful submission can be determined using a linear function that incorporates three types of scores mentioned earlier:

$$p(S) = w_c * f_c(S) + w_e * f_e(S) \qquad \ldots(3)$$

In this equation, the weights (w's) are assigned based on the importance of each score type. These weights are learned through a linear regression model. To ensure that the weights meet a constraint, their sum is set to 1, i.e., $w_c + w_e = 1$ [14]. It is important to note that the performance p(S) falls within the range of 0 and 1 [14]. To suggest next programming task, the performance will be compared with different levels of performance value.

III. Evaluation

Evaluation can be done based on task submission results. Here we are discussing each test case of programming task submission evaluation.

A. Code did not pass all the test cases

This situation can occur when there is an error in the code or when the program's output does not match the expected result. In such cases, the learner is typically given a set number of attempts to successfully complete the code. If the remaining trials reach zero, the learner will receive a code complexity level that is less challenging for the same task.

However, if the learner has already reached the lowest complexity level, c_1, for the task, it becomes mandatory to complete that task and achieve an acceptable level of complexity c_i. Here, i is an acceptable level of complexity set by the instructor, which is $\geqslant 1$ and $\leqslant n$, where n is the highest level of complexity same for each topic.

B. Code passed all the test case

If code gets passed from a predefined set of test cases successfully, then the final performance calculation will start. For measuring the performance of the learner, the instructor can set a benchmark value between 0 to 1, based on which proper decision can be made. These benchmarks are P_{accept}, P_{min}, and P_{max}. P_{accept} is the minimum acceptance level, if the performance score, p(S), is less than it, then the learner will be provided a lower complexity task on the same topic. P_{min} is the next acceptance level, if the performance score, p(S), is more or equal to P_{min}, then only the learner will be able to move to n

Here, if the performance value is more than or equal to P_{min}, less than P_{max} and the current complexity level is greater than or equal to c_i, where then the next topic task with the same complexity can be given to the learner. If the performance value is more than P_{max} then the next topic task with a higher complexity level will be assigned to the learner.

Table-1 Code did not pass all test cases

Code Submission Status	**Complexity Level, c_i**	**Number of remaining trials t_r**	**Model Decision**
Did not pass all test cases	$c_i, i\neq 1$	$t_r > 0$	Same topic, Same task with the same Complexity level c_i
	$c_i, i>1$	$t_r = 0$	Same topic, New task with Complexity level c_{i-1}
	$c_i, i=1$	$t_r = 0$	It mandatory to complete the task or another task for the same topic with the same complexity level c_i may be assigned.

Table-2 Code passed all test cases

Code Submission Status	**Task Level, T_k**	**Complexity Level, c_i**	**Performance Score, p(S)**	**Model Decision**
Passed all test cases	$T_k,\ 1\leq k<n$	$c_i, i=1$	$p(S)<P_{accept}$	Same task with the same Complexity level c_i
		$c_i,\ 1<i\leq n$	$p(S)<P_{accept}$	Next task with Lower Complexity level c_i
		$c_i,\ 1\leq i\leq n$	$P_{accept} \leq p(S)<P_{min}$	Same task with Higher Complexity level c_{i+1}
		$c_i,\ 1\leq i<n$	$P_{min} \leq p(S)<P_{max}$	Next task with the same Complexity level c_i
			$p(S) \geq P_{max}$	Next task with Higher Complexity level c_{i+1}
	$T_k, k=n$	$c_i,\ 1\leq i<n$	$P_{min}\leq p(S)<P_{max}$	Next Topic, New task with Complexity level c_i
			$p(S) \geq P_{max}$	Next Topic, Next task with Higher Complexity level c_{i+1}
		$c_i,\ i=n$	$p(S) \geq P_{max}$	Next Topic, Next task With the same Complexity level c_i

The model decision is based on various factors, including task level (T_k), complexity level (c_i), performance score (p(S)), and model decision. When all test cases are passed, indicating successful completion of a task (T_k) with complexity level c_i and a performance score below P_{accept}, the student proceeds to the same task with the same complexity level. If the performance score remains below P_{accept} for subsequent tasks with the same complexity level (c*i*) but different levels (i), the student progresses to the next task with a lower complexity level. Conversely, if the performance score falls between P_{accept} and P_{min}, the student continues with the same task but at a higher complexity level (c_{i+1}). If the score exceeds P_{max}, the student advances to the next task, either with the same or higher complexity level, depending on the task level and completion status. Finally, if the performance score surpasses P_{max} at the final task level (k=n), the learner moves on to the next topic, tackling a new task with the corresponding complexity level (c_i).

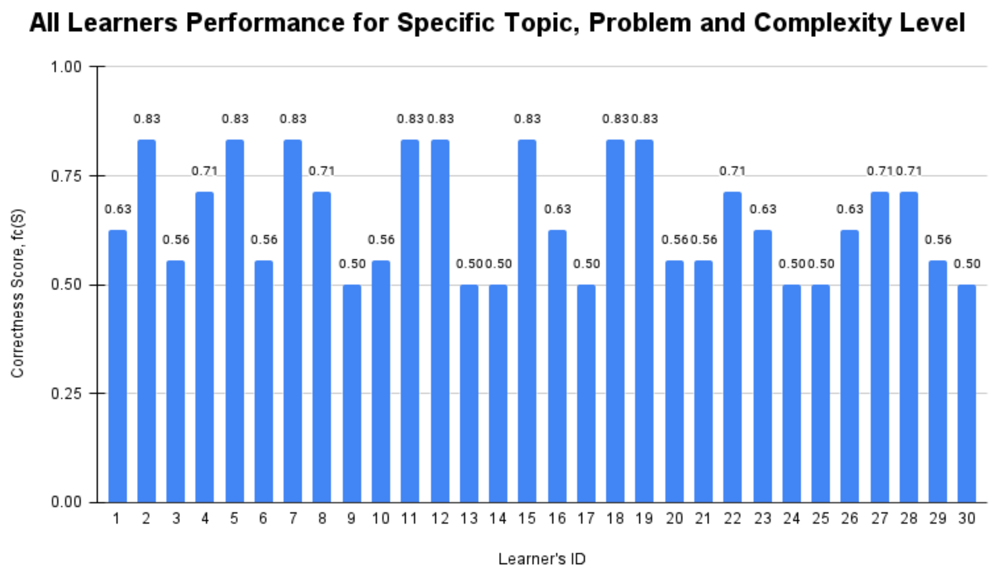

Fig. 1 – Correctness Score, $f_c(S)$

In Fig. 1, we can see the correctness score, $f_c(S)$, of all learners for the task with the same topic and complexity level. Instructors can get valuable insights from Fig 1., like the number of unsuccessful attempts to complete the task. This information becomes more useful for the instructor in teaching learners how to overcome logical errors.

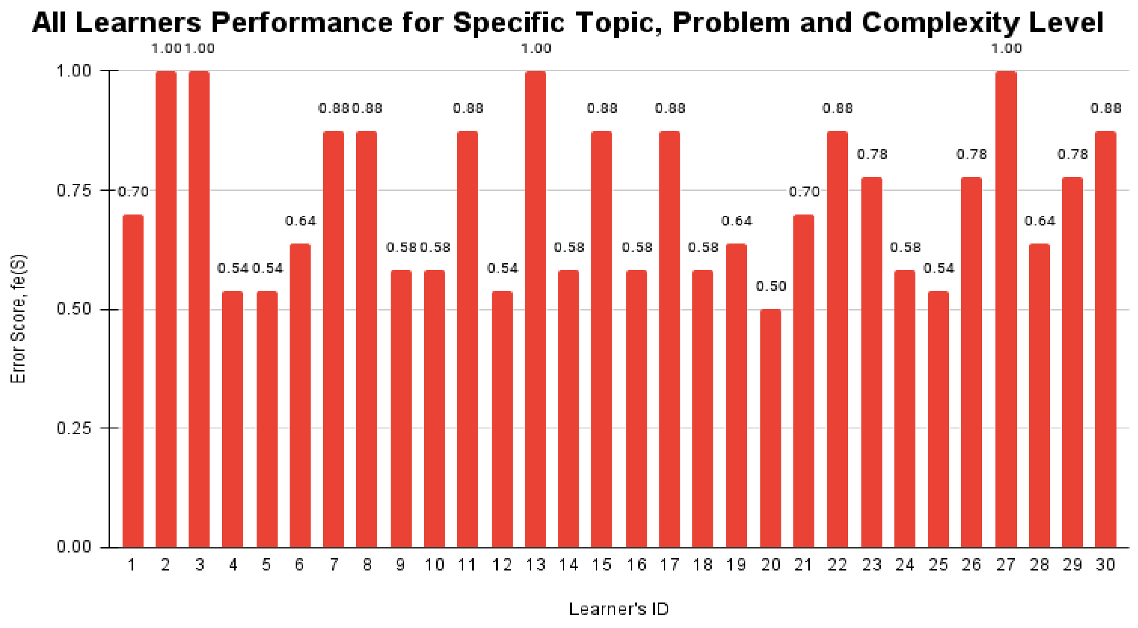

Fig. 2 – Error Score, $f_e(S)$

In Fig. 2, we can see all learners' error scores, $f_e(S)$, for the task with the same topic and complexity level. Instructors can learn about the types and number of errors, specifically syntactical errors, that learners face during solving tasks. This information helps the instructor in teaching programming more effectively.

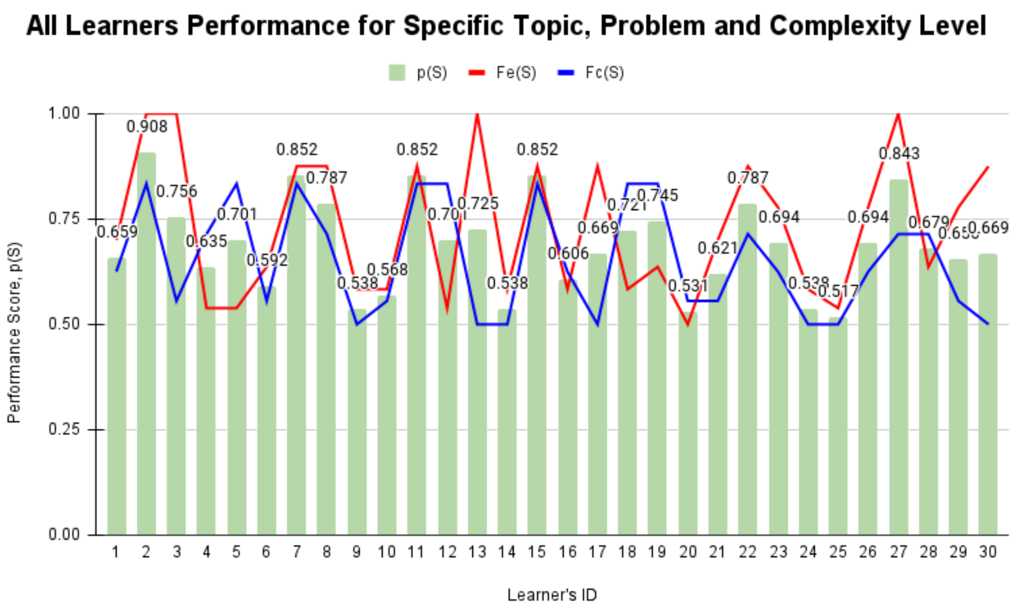

Fig. 3– Performance Score, p(S), for all 30 learners

Fig. 3 shows the performance score, p(S), calculated using equation (3), relation with Correctness Score, $f_c(S)$, and Error Score, $f_e(S)$.

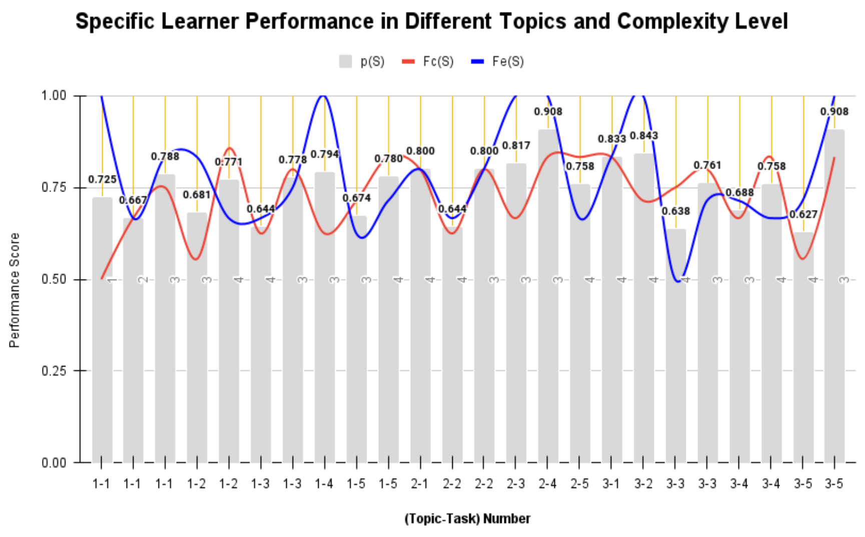

Fig. 4– Specific Student's Performance in all topics

In Fig. 4, we can see the performance score, p(S), of one specific learner that shows how the learner performed in the course. It depicts the performance of learners in specific topics at different complexity levels.

IV. Conclusion

This paper focuses on the issue of automatic code assessment, aiming to assess both learner's comprehension of a topic and their ability to tackle complex programming tasks. By utilizing equation (3) and establishing an acceptable dynamic average complexity level, denoted as "i," we can assign varying values to wc and we. This allows for dynamic complexity values of P_{accept}, P_{min} and P_{max}, tailored to each group of learners. Consequently, this approach enables instructors to gain insights into the learning difficulties faced by individual learners.

In the future, we plan to conduct experiments with various groups of learners studying the basic programming language C. These experiments will involve a specific topic and a list of tasks related to C programming. By applying our model to these different sets of learners, we aimed to gather valuable insights and evaluate the model's effectiveness in enhancing their understanding and performance in programming. Our aim is to construct a recommender system assisted by data gathered during this research. By analyzing the performance of new learners in their initial tasks, the recommender system will proficiently suggest suitable tasks aligned with comparable programming skills and performance levels.

References

[1] Nguyen, A., Piech, C., Huang, J., & Guibas, L. (2014). Codewebs. https://doi.org/10.1145/2566486.2568023

[2] Rubio, M. A. (2020). Automated Prediction of Novice Programmer Performance Using Programming Trajectories. In Lecture notes in computer science (pp. 268–272). https://doi.org/10.1007/978-3-030-52240-7_49

[3] Hosseini, R., Brusilovsky, P., Yudelson, M., & Hellas, A. (2017). Stereotype Modeling for Problem-Solving Performance Predictions in MOOCs and Traditional Courses. https://doi.org/10.1145/3079628.3079672

[4] Bresfelean, V. P., Bresfelean, M., Ghisoiu, N., & Comes, C. A. (2008). Determining students' academic failure profile founded on data mining methods. https://doi.org/10.1109/iti.2008.4588429

[5] Ahmed, U. Z., Sindhgatta, R., Srivastava, N., & Karkare, A. (2019). Targeted Example Generation for Compilation Errors. https://doi.org/10.1109/ase.2019.00039

[6] Murphy, C., Kaiser, G., Loveland, K., & Hasan, S. (2009). Retina. SIGCSE Bulletin, 41(1), 178–182. https://doi.org/10.1145/1539024.1508929

[7] Alalawi, K., Athauda, R., & Chiong, R. (2023). Contextualizing the current state of research on the use of machine learning for student performance prediction: A systematic literature review. Engineering Reports, 5(12). https://doi.org/10.1002/eng2.12699

[8] 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(3), 1–19. https://doi.org/10.1145/3277569

[9] Restrepo-Calle, F., Ramírez-Echeverry, J. J., & Gonzalez, F. A. (2018). UNCODE: Interactive System For Learning And Automatic Evaluation Of Computer Programming Skills. EDULEARN Proceedings. https://doi.org/10.21125/edulearn.2018.1632

[10] Ala-Mutka, K. M. (2005). A Survey of Automated Assessment Approaches for Programming Assignments. Computer Science Education, 15(2), 83–102. https://doi.org/10.1080/08993400500150747

[11] Glassman, E. (2016). Clustering and visualizing solution variation in massive programming classes. Massachusetts Institute of Technology. Retrieved May 29, 2024, from http://up.csail.mit.edu/other-pubs/elg-thesis.pdf

[12] Rivers, K., & Koedinger, K. R. (2015). Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. International Journal of Artificial Intelligence in Education, 27(1), 37–64. https://doi.org/10.1007/s40593-015-0070-z

[13] Watson, C., Li, F. W., & Godwin, J. L. (2013). Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. https://doi.org/10.1109/icalt.2013.99

[14] Parihar, S., Dadachanji, Z., Singh, P. K., Das, R., Karkare, A., & Bhattacharya, A. (2017). Automatic Grading and Feedback using Program Repair for Introductory Programming Courses. https://doi.org/10.1145/3059009.3059026

[15] Devi, S., Selvam, K., & Rajagopalan, S. (2011). An abstract to calculate big O factors of time and space complexity of machine code. https://doi.org/10.1049/cp.2011.0483

[16] Robins, A. V. (2019). Novice Programmers and Introductory Programming. In Cambridge University Press eBooks (pp. 327–376). https://doi.org/10.1017/9781108654555.013