

Programming Question Generation: An Automated Methodology for Generating Novel Programming Assignments with Varying Difficulty Levels

Dhanya E, Nikhila KN

¹*International Institute of Information Technology Bangalore, India.*

Abstract

A comprehensive programming course requires a clearly delineated collection of programming activities that can help students improve their programming skills. Instructors devote considerable time to preparing such problems and their corresponding solutions. They often resort to a strategy of using previously developed materials from prior semesters and drawing upon online resources to facilitate the preparation of course materials. We are presenting a methodology for generating practice exercises that cater to students with diverse levels of difficulty. The proposed approach is beneficial for educators in developing instructional resources and personalised assessments customised to individual students' programming proficiency.

Keywords

Intelligent tutoring systems, Computing Education, Natural Language Generation, Resource Generation

1. Introduction

Generating programming questions automatically for programming assignments in intelligent tutoring systems are a rapidly expanding area and has the potential to improve significantly computer science education. Programming assignments are commonly used to assess a student's comprehension and application of programming concepts.

It is necessary for the instructor to prepare programming assignments with varying levels of difficulty. At the beginning of a course, the assignments should be basic and solvable for a beginner. The complexity of the assignments should then gradually increase as the students progress through the course, in order to provide students with an adequate challenge, appropriate for each skill level. If assignments are too difficult right at the beginning, students may get demotivated, which could lead to ethically dishonest behaviour as they try to meet assignment deadlines [? ?].

In this context, it is useful to personalise programming exercises to the individual knowledge levels of students. This methodology guarantees that every student is presented with questions that are appropriate for their existing programming proficiency, thereby facilitating more efficient practise and fostering confidence. Thus, this customised approach facilitates the enhancement of their programming proficiency. In addition, furnishing distinctive question sets


Woodstock'21: Symposium on the irreproducible science, June 07–11, 2021, Woodstock, NY

✉ dhanya.eledath@iiitb.ac.in (D. E); nikhila.kn@iiitb.ac.in (N. KN)

ORCID 0000-0001-5954-8554 (N. KN)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

to individual students serves as a deterrent against the tendency to plagiarise from their peers' submissions [? ? ?]. The learning environment fosters independent thinking and problem-solving skills among students by discouraging the use of repetitive or identical exercises.

However, creating effective programming assignments and reference solutions can be tedious and time-consuming, especially when designing questions with varying levels of difficulty. In addition, the prevalence of academic dishonesty through plagiarism has increased as a result of the reuse of programming assignment questions across multiple semesters and the availability of code on the internet, as indicated by research findings [?]. This highlights the importance of an automated question generation tool that can generate unique programming assignment questions, thereby reducing the likelihood of plagiarism and encouraging originality of student responses.

Automated question generation, which generates high-quality programming questions for students to work on, can be an effective solution to this issue. This method saves instructors time and effort as they do not have to manually create each programming question.

Furthermore, tailoring the questions to the student's level of knowledge provide each student with an appropriate level of challenge, ultimately resulting in a fair and impartial evaluation of their knowledge.

2. Relatedwork

In each programming course, programming assignments are a crucial assessment method used by instructors to evaluate student's knowledge and understanding of programming concepts [?]. According to the literature, the significance of programming assignments in programming courses has been studied for three decades [?]. And it is clear that setting assignments with varying degrees of difficulty is essential for maintaining student motivation to learn [?]. A well-structured programming assignment is required to help students develop programming skills and a better understanding of the problem. There exist various challenges in the preparation of assignment problems and this review aims to emphasize the importance of assignment question preparation with varying levels of difficulty.

Gondim et al. [?] proposed a new teaching approach for introductory programming (CS1), combining Problem-Based Learning (PBL) with the flexibility of Extreme Programming (XP) to produce a more collaborative, challenging, and dynamic learning environment. This strategy aims to improve student's abilities by applying software engineering best practices and enhancing code quality. To facilitate the XP-based problem-solving process, the TaskBoard application was developed to help student groups in the creation, management, and persistence of solutions and related artifacts. This methodology introduces a novel approach to programming education that could improve student learning outcomes by fostering a greater active engagement with programming concepts.

However, the challenge of designing programming assignments that are suitable for students with diverse levels of prior experience cannot be underestimated. As the range of prior experiences within a cohort becomes more diverse, instructors struggle to produce learning resources at an appropriate level of difficulty for individuals [?]. To address this challenge, Flynn et al. [?] proposed a novel open-source tool for delivering *faded parsons* problems. Parsons problems

are a popular programming exercise that presents learners with the lines of code for a solution in a scrambled order. Faded parsons problems present some lines of code with missing portions that students must fill in. The study evaluated the relative difficulty of three distinct fading strategies and found that fading conditional statements greatly increases the difficulty of *faded parsons* problems. The study's results could inform future automated approaches for adapting to the difficulty of such problems.

The majority of instructors are willing to share the results of the time they invest in designing programming assignments. However, most of the programming assignments are shared ad hoc through informal channels [?]. To aid instructors in locating and sharing such materials, Stephen et al. [?] proposed a standard format for sharing assignments. This format is simple for instructors to create, is extensible and flexible enough to accommodate assignments written in any programming language and at any level of expertise, supports appropriate metadata, and is easily manipulated by software tools. As more instructors employ automated grading tools to evaluate student submissions, such an interchange format could lead to a community practice of sharing resources in a manner that overcomes existing barriers to such reuse.

Sami Sarsa et al. have explored the potential of automated generation of programming assignment problems, sample solutions, test cases, and explanations using large language models. In their study [?], they have detailed the methodology they have used, which involves utilizing the natural language generation capabilities of pre-trained large language models, specifically Open-AI's Codex. Their approach takes inputs such as keywords, problem statements, sample solutions, and tests for specific programming concepts, and generates new problem statements, keywords, tests, and sample solutions in natural language.

To evaluate the effectiveness of their approach, the researchers conducted both quantitative and qualitative studies. The quantitative study aimed to identify the novelty of the generated problems and the usefulness of the method. The researchers evaluated sensibleness, novelty, and readiness to use the generated problems and solutions as they were, as well as assessing whether the generated problems matched with the input priming concepts provided.

The qualitative study involved gathering feedback from a group of experts in programming and natural language generation. The experts were asked to evaluate the quality of the generated problems, sample solutions, test cases, and explanations, and to provide feedback on the overall effectiveness of the approach.

The results of the studies showed that the automated generation of programming assignment problems, sample solutions, test cases, and explanations using large language models is a promising approach. The generated problems were found to be sensible, novel, and ready to use, and they matched well with the input priming concepts. The feedback from the experts indicated that the generated problems, sample solutions, test cases, and explanations were of high quality and could be used in programming education and assessment.

Generation of code tracing problems which is also beneficial in programming courses is well studied [?]. The authors in the paper introduced an automated generation of code tracing problems in varying complexity and studied the method's effectiveness through its usage in the programming course.

From the literature, it is evident that well-structured programming assignments with varying levels of difficulty are necessary for students to understand programming concepts and develop programming skills.

3. Proposed Method

3.1. Problem definition

In this paper, we propose a methodology that can be used to generate novel programming problems, the corresponding reference solutions and input-output specifications. The problems generated using this methodology can be used by instructors to release assignments without significant investment of time and effort. The use of a large language model holds the potential in facilitating the creation of reference solutions for the questions generated. At present, our primary focus is on the creation of the problem set.

Given a programming concept and its complexity, the model outputs a programming description, consisting of the question and sample input-output set related to this question. Below, we show an example programming problem description generated by the model when fed with the inputs: programming concept - *recursion* and the level of difficulty - *easy*.

Concept: Recursion

Difficulty: Easy

Problem Description:

Write a generator function that returns a generator object which yields the fibonacci sequence. The fibonacci sequence is defined by the relation $X_n = X_{n-1} + X_{n-2}$.

The first few numbers of the series are 0, 1, 1, 2, 3, 5, 8, 13.

Example 1:

Input: callCount = 5

Output: [0, 1, 1, 2, 3]

3.2. System architecture

In our work, we leverage the knowledge gained by pre-trained Large Language Models (LLM) (trained on large amounts of data) for a specific downstream task. The downstream task in our case is to generate programming descriptions related to a particular programming concept.

Large language models are capable of doing a variety of language-related activities, including conversing, translating languages, answering questions, composing essays, summarising data, and summarising knowledge. We use this potential of Large language models in our automated question generation tool to assist instructors. There are several large language models available such as T5 [?], OpenAI's GPT-3.5 [?].

The approach that we propose is of two kinds:

1. Use large language models as it is without fine-tuning. We make use of GPT-3.5 [?] in our first phase of experimentation as it best suits our problem definition.
2. In our second phase, we use pre-trained models like T5 [?] as our upstream model and further fine-tune this model using our task-specific dataset (as shown in Figure 1). The fine-tuned model is later used to generate programming descriptions, given a programming concept and difficulty level.

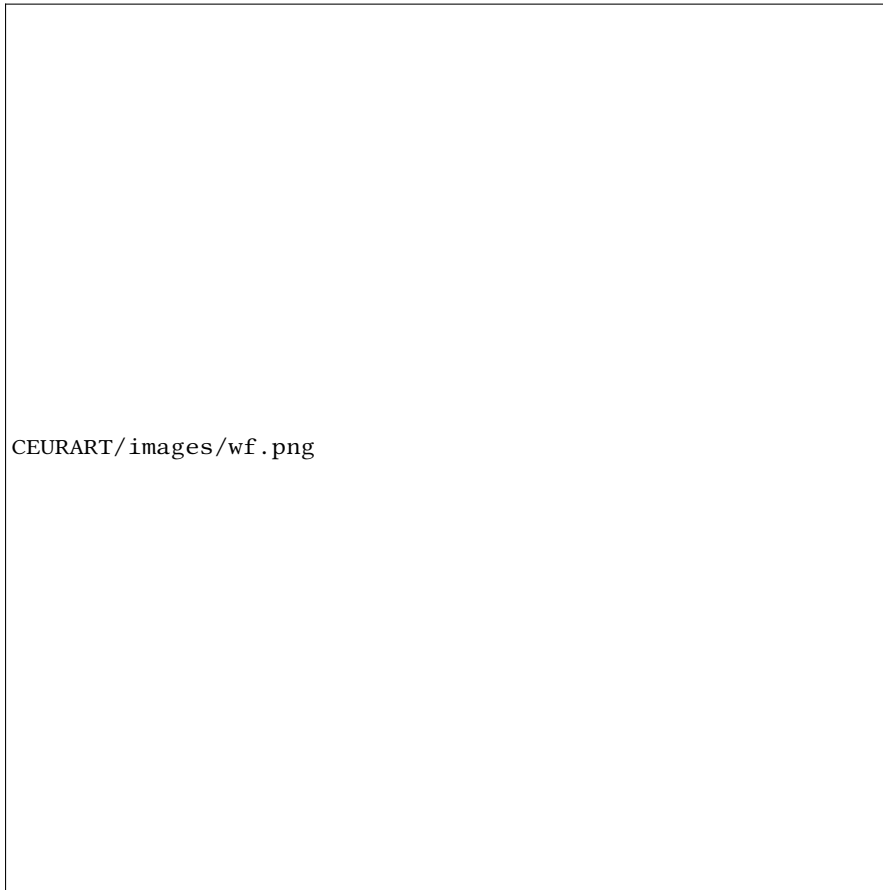


Figure 1: Pipeline showing the training and fine-tuning stage of our proposed system.

3.3. Dataset details

The datasets used in the experiments were created by collecting programming questions and their corresponding tags from HackerRank and LeetCode [? ?]. We collected a total of 596 problem sets from Lead Code and 1600 problem sets from HackerRank. Additionally, we are presently considering strategies to extend and enhance these datasets for subsequent investigations, with the objective of constructing a moderately robust framework for question generation. The problem description collected from HackerRank and LeetCode platforms includes additional information. The specifications of the expected implementation include input-output constraints and running time constraints etc. The information mentioned earlier has not been used for the purpose of training our model. Consequently, it was necessary to perform preprocessing on the problem description that was downloaded. The sole components used in our approach are the problem description, example input and output, difficulty level, and concept tags. All other additional information has been eliminated to standardise the format of all problem sets within the dataset.

3.4. Implementation and experiments

This research involves developing a model to generate programming assignment questions and analysing its output to assess whether it meets the desired criteria, including novelty and difficulty level.

Recent research efforts have however shown that it is possible to fine-tune LLMs without updating all of the parameters [? ? ? ?]. These techniques are useful for producing outcomes that are as close to the model with all the parameters set as possible. Any of the following approaches can be taken into consideration for the model's fine-tuning for the experiments in this project.

- Prompt Tuning - Either Hard prompt tuning or soft prompt tuning
- Prefix Tuning

In the initial phase of our experimentation, we used OpenAI's GPT-3.5 model through their API [?]. The model exhibits the capacity to produce natural language output via prompt tuning. The experiments were conducted using prompt tuning techniques, with a prescribed *stop sequence* for the model set at a maximum of 75 words. The generative capacity of the model is regulated by the *temperature* parameter. Throughout the experiment, the model's temperature value remained at 0.7. Figure 2 presents the sample output produced by the model. The prompt has been designed to accept programming concepts of varying levels of complexity as input, which are then processed by the API to generate a corresponding programming problem description.

Large language models are trained on millions of text corpora and are capable of generating texts. Using these models for a specific task without fine-tuning can compromise the expected accuracy [?], whereas the process of fine-tuning a model by updating all of its parameters is expensive and necessitates a significant amount of data. Therefore, in the first phase of the implementation process we resort to a non-finetuning approach.

In our second approach, we fine-tune an existing large language model, *T5*, with the aim of generating novel programming questions. The fine-tuning process entails leveraging features such as problem title, tags(which represent programming concepts), and difficulty level, with the programming description serving as ground truth for training purposes.

4. Conclusion

This paper presents the significance of incorporating programming practice questions of diverse difficulty levels in a programming course. Furthermore, experiments were conducted regarding the natural language generation ability of the GPT model. The main feature of our methodology resides in its capacity to generate questions of varying degrees of complexity. The problems that are generated have demonstrated their usefulness for both students and instructors, as they facilitate the improvement of programming skills for students and support instructors in developing course materials and assignment problems. As a continuation of this work, we aim to expand our experimentation to include an alternative open-source large language model and compare its performance to that of the GPT model, which serves as our baseline. In addition,



Figure 2: Output generated from the model *gpt-3.5-turbo*.

we would like to conduct a user study involving domain experts and students to investigate the efficacy of the generated programming questions.

References

- [] A. Lipson, N. McGavern, Undergraduate academic dishonesty at mit. results of a study of attitudes and behavior of undergraduates, faculty, and graduate teaching assistants. (1993).
- [] E. Roberts, Strategies for promoting academic integrity in cs courses, in: 32nd Annual Frontiers in Education, volume 2, 2002, pp. F3G–F3G. doi:10.1109/FIE.2002.1158209.
- [] M. Dick, J. Sheard, C. Bareiss, J. Carter, D. Joyce, T. Harding, C. Laxer, Addressing student cheating: Definitions and solutions, in: Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education, ITiCSE-WGR '02, Association

for Computing Machinery, New York, NY, USA, 2002, p. 172–184. URL: <https://doi.org/10.1145/960568.783000>. doi:10.1145/960568.783000.

- [] R. C. Hollinger, L. Lanza-Kaduce, Academic dishonesty and the perceived effectiveness of countermeasures: An empirical survey of cheating at a major public university, *NASPA Journal* 46 (2009) 587–602. URL: <https://doi.org/10.2202/1949-6605.5033>. doi:10.2202/1949-6605.5033. arXiv:<https://doi.org/10.2202/1949-6605.5033>.
- [] I. Albluwi, Plagiarism in programming assessments: A systematic review, *ACM Trans. Comput. Educ.* 20 (2019). URL: <https://doi.org/10.1145/3371156>. doi:10.1145/3371156.
- [] H. W. A. S. Gondim, A. P. L. Ambrósio, F. M. Costa, Taskboard - using xp to implement problem-based learning in an introductory programming course, in: A. Sillitti, O. Hazzan, E. Bache, X. Albaladejo (Eds.), *Agile Processes in Software Engineering and Extreme Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 162–175.
- [] M. C. Linn, M. J. Clancy, The case for case studies of programming problems, *Commun. ACM* 35 (1992) 121–132. URL: <https://doi.org/10.1145/131295.131301>. doi:10.1145/131295.131301.
- [] F. Fromont, H. Jayamanne, P. Denny, Exploring the difficulty of faded parsons problems for programming education, in: *Proceedings of the 25th Australasian Computing Education Conference, ACE '23*, Association for Computing Machinery, New York, NY, USA, 2023, p. 113–122. URL: <https://doi.org/10.1145/3576123.3576136>. doi:10.1145/3576123.3576136.
- [] S. H. Edwards, J. Börstler, L. N. Cassel, M. S. Hall, J. Hollingsworth, Developing a common format for sharing programming assignments, *SIGCSE Bull.* 40 (2008) 167–182. URL: <https://doi.org/10.1145/1473195.1473240>. doi:10.1145/1473195.1473240.
- [] S. Sarsa, P. Denny, A. Hellas, J. Leinonen, 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, ICER '22*, Association for Computing Machinery, New York, NY, USA, 2022, p. 27–43. URL: <https://doi.org/10.1145/3501385.3543957>. doi:10.1145/3501385.3543957.
- [] E. Stankov, M. Jovanov, A. Madevska Bogdanova, Smart generation of code tracing questions for assessment in introductory programming, *Computer Applications in Engineering Education* 31 (2023) 5–25. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22567>. doi:<https://doi.org/10.1002/cae.22567>. arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/cae.22567>.
- [] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, 2020. arXiv:1910.10683.
- [] OpenAI, Language models are few-shot learners (2021). URL: <https://arxiv.org/abs/2105.14103>. arXiv:2105.14103.
- [] Hackerrank, <https://www.hackerrank.com/>, ???? Online; accessed [24-02-2023].
- [] Leetcode, <https://leetcode.com/>, ???? Online; accessed [24-02-2023].
- [] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, *CoRR abs/1810.04805* (2018). URL: <http://arxiv.org/abs/1810.04805>. arXiv:1810.04805.
- [] V. Lialin, V. Deshpande, A. Rumshisky, Scaling down to scale up: A guide to parameter-

efficient fine-tuning, 2023. [arXiv:2303.15647](#).

- H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, Llama: Open and efficient foundation language models, 2023. [arXiv:2302.13971](#).
- R. Zhang, J. Han, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, P. Gao, Y. Qiao, Llama-adapter: Efficient fine-tuning of language models with zero-init attention, 2023. [arXiv:2303.16199](#).
- X. L. Li, P. Liang, Prefix-tuning: Optimizing continuous prompts for generation, 2021. [arXiv:2101.00190](#).
- S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, R. Liu, Plug and play language models: A simple approach to controlled text generation, 2020. [arXiv:1912.02164](#).