Utilizing Large Language Model for Programming Course Exercise Generation

Erkki Kaila[1][0000−0002−2407−9492], Juuso Rytilahti[1][0009−0002−8269−5645], William Lempinen[1], and Luuka Lindgren[1]

University of Turku, Finland {ertaka, jubery, wislem, luslin}@utu.fi
http://www.utu.fi

Abstract. Large language models (LLMs) are potentially powerful tools for automating educational tasks. In this paper, we observe two use cases of LLMs related to introductory programming education. In the first case, we created an LLM-based tool for creating variations of existing exercises. In the second case, we used LLM for generating the unit tests and good-quality feedback for students' answers to programming exercises. Both approaches were studied by gathering data from two instances of a large introductory programming course. Our results indicate, that both approaches were successful. In addition to discussing the results, we discuss the insights gained, the identified use cases, and the significance of the rapid progress the LLMs have on programming education.

Keywords: LLM · Programming education · Exercises · Variations · Automated testing.

1 Introduction

Large language models (LLMs) can potentially help teachers in various education-related tasks. This is especially true in larger courses. In this paper, we discuss the use of LLMs in exercise generation in the context of a large introductory programming course. The first problem is related to renewing the courses: it is usually necessary to re-write at least a part of course exercises frequently to avoid students using the old answers. However, coming up with new exercises that require the same skills can be problematic. The second problem is related to automatic assessment of the student answers. Writing tests that assess the student code from all necessary aspects, and which provide useful feedback immediately, can also take a lot of time.

Hence, the paper tries to answer two research questions:

RQ 1: Is it possible to use an LLM to generate variations of the existing exercises with the same quality as the original ones?
RQ 2: Is it possible to use an LLM to automatically create tests for introductory programming exercises?

Both research questions are studied separately, but the context is the same: a large introductory university-level programming course. We have previously [21] discussed programming exercise generation with three different approaches:

- *Theme Injection*: Model generates a programming exercise from a specific topic with a specific theme without an original exercise as input.
- *Variation Generation*: The model produces a similar exercise as the input, only changing the context or the theme of the exercise.
- *Hybrid Exercise Generation*: Model attempts to combine existing exercises into new exercises using a different context, but utilizing the skills required in the exercises given in the input.

In this paper, we are mainly focusing on the second one, but also describe the use of theme injection and its role in the variation tool we have built.

The rest of the paper is structured as follows: in Section 2, we present related scientific literature, focusing on LLM use in introductory programming, exercise generation, and testing. In the third section, the context and methodology are presented. In Sections 4 and 5, we present the setup, tool generation, and the results of our studies. The results are discussed in Section 6 and concluded in Section 7.

2 Related Work

2.1 LLM's in Programming Education

At the time of writing this, Large Language Models have been in general use for no more than a little over two years. Still, there are already many examples of utilizing them in introductory programming education [1]. Authors of [8] used an LLM to generate working examples for a programming course. Such examples can be highly useful for learning but are usually quite laborious to create. Similarly, [22] presents an approach where they used an LLM and customized role-based agents for generating high-quality programming projects for educational purposes. [14] highlight the importance of researching the use of LLM-enhanced tools over extended periods; with this in mind, in this article, we have collected data from multiple instances of the same course.

[12] present a study where they compared line-by-line code explanations made by LLM to expert-made explanations. They conclude, that while the explanations are lexically and semantically similar, they are less readable, which lowers the usability. Still, the LLM-generated explanations are a lot closer to expert-produced than the ones written by students. In [6] the authors provide a multifaceted approach to using LLM in introductory programming, as they use chatbots to generate, explain, and simplify code. According to the authors, the approach worked especially well with smaller problems. Another interesting approach is reported by [20], where the LLMs are used to produce hints that help the students overcome problems when writing the programs.

2.2 Exercise Generation

ExGen [23] is an exercise generator capable of producing Python exercises for introductory programming courses. It maintains a set of "seed exercises" in the

user's private database and uses a cloud-based LLM to generate new exercises. It then filters the most suitable exercises from the generated candidates. Authors of [15] tested two different LLM models for generating programming exercises for higher education, and emphasized the critical role of human supervision in selecting and editing the generated exercises. Generating exercises automatically can enable new approaches, such as contextually personalized programming exercises [13], which adapt to students' interests or needs, but also require new skills, such as prompt writing for solving programming-related problems [2].

2.3 Exercise Assessment and Feedback

In programming courses, it is quite typical to use some kind of form of automated testing of exercises (see e.g. [24], [18]). This is done to prevent time needed for assessing the exercises and to provide automated and immediate feedback for students, which enables them to fix and re-submit their solution without a wait. Feedback generation can also be enhanced with LLM. [10] developed CodeAid, which "answers conceptual questions, generates pseudo-code with line-by-line explanations, and annotates student's incorrect code with fix suggestions". They conclude the pilot usage by providing four suggestions for developing such tools, including exploiting AI's unique benefits and avoiding giving direct answers. The evaluation of LLM's feedback capabilities by [4] was, however, not very positive as they state that the accuracy of LLM-generated feedback was less than 50%.

3 Methodology

3.1 Context

The exercise generation was tested and observed in an introductory programming course at the University of Turku, Finland. Python is used as a programming language of the course, and no previous programming knowledge is required. The course is a typical CS1 course, covering the topics of imperative paradigm: variables, conditional statements, loops, and functions. Additionally, basic data structures (lists, tuples, and dictionaries), file operations, and the use of external modules are discussed. The course is divided into seven sections, and with one section covered per week, the total length is eight weeks – the last week is dedicated to the exam. The course can be taken fully online except for the final exam, which is taken supervised at the University premises.

Most of the exercises in the course are coding tasks, where the students need to write a program according to the given specifications. At the end of each week, there is typically a larger task, which consists of several exercises. Each week concludes with a feedback survey where the students are asked what they learned, what things remain unclear, and how difficult they find the material that week. The weekly exercises are done in ViLLE [11], which is a web-based, collaborative learning tool. The exercises are automatically assessed and provide immediate feedback after the answer is submitted. The students have 9 days to complete each round of exercises, and during that time, each exercise can be tried as many times as needed.

3.2 Participants

The course is primarily aimed at computer science majors, but it also serves as a minor study for other majors in the university. The course is quite large: of the two instances related to this paper, the 2023 instance had a total of 580 students, and the 2024 instance had a total of 557 students. The students are mostly first-year students with little to no previous experience in programming. All personal data was removed before analysis, meaning that the analyzed data was fully anonymous.

3.3 Data Collection and Analysis

ViLLE automatically collects data from each submission, including the score achieved and the time spent on doing the task. ViLLE also records the number of submissions each student spent on each submission; this data is collected mainly for research or course development purposes, as the number of submissions does not affect student scores. The dataset used for the analysis contained each student's best score for each exercise they had completed, the total amount of time in seconds they had spent doing each exercise, and the total number of submissions they had done for each exercise. Hence, the average scores are quite high, as it is quite typical for the students to keep on working on the exercise until they get it correct.

Additional survey data was used for analyzing the first research question. To find out whether the students could tell the AI-generated exercises from human-generated ones, one additional question was included in the feedback survey in the final week. All survey data was also collected via ViLLE, and to encourage students to answer the surveys, they could collect some points for answering them. Survey data was not used to collect feedback from the exercises utilizing the modified automated testing (RQ2), as we felt that students could not provide meaningful feedback without the possibility to compare them to older versions.

4 Exercise Variation Tool

4.1 Background

As most teachers agree, preparing the materials for a course is a huge task. While some of the material can usually be recycled in upcoming years with little updates, some materials need constant updating or even rewrites. Different course tasks are a typical example of this. If the same assignments are used for several instances, the correct answers can likely be found in some repository, discussion forum, or other location accessible by students. However, re-designing and re-writing an entire set of exercises for each instance can be too big of a workload to be completed regularly.

4.2 Using ChatGPT to Generate Exercise Variations

Often, it is not necessary to create completely new exercises. Instead, creating a variation of an existing exercise may be enough. The variation can be of a different theme or provide a different initial setup, while still maintaining the basic idea and requiring the same set of skills to solve. This also means, that the accompanying materials do not probably require too much work, as the exercise is fundamentally similar. However, while a similar idea is kept, the existing solutions do not work, making the exercise "novel enough".

With this in mind, we decided to utilize a large language model called ChatGPT for generating variations of existing programming tasks. After testing the generation with individual prompts, we decided to build a simple tool for making the process more straightforward. OpenAI provides an API [19] (Application Program Interface) which can be used to access ChatGPT from other programs. This was used to build a prototype application that could be used to generate the variations. The application is displayed in Fig. 1.

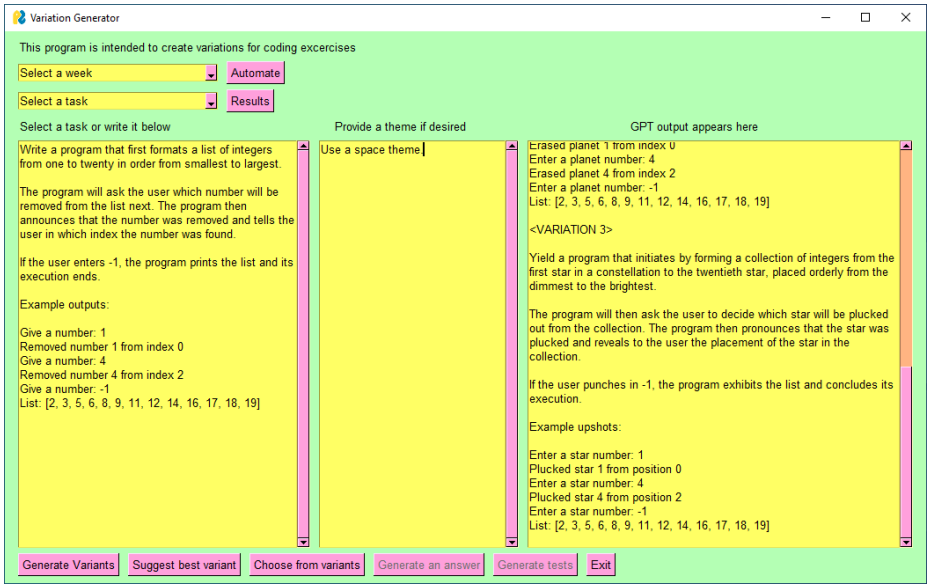

Fig. 1. The main view of the prototype application.

The programming exercises from the course were pre-loaded into the application enabling the user to quickly generate exercise variations. Additionally, new tasks can be entered by using the leftmost editor window. In the middle frame, the user could provide a theme (such as "space theme" or "medieval setting"), to support a comprehensive theme of a larger section or to provide fun alteration. The tool automatically provides three variations of the exercise, and the user can select the preferred one to further edit or use it as it is.

4.3 Results

In the second phase, we decided to test out the variations in the course. We picked up three exercises from the final week of the course to be replaced by the variations generated with the prototype tool. The original exercises and the generated variations are displayed in Table 1.

Table 1. The original tasks and the variations generated by the variation tool.

Original Exercise	Generated Variation
Generate a list of lengths based on a given list of strings by using list comprehension.	Generate a list of lengths based on a list of reports by using list comprehension.
Generate a list of students who passed the course based on a given list of students and their exam scores.	Generate a list of employee bonuses based on a given list of employees and their yearly sales.
Order a list of football teams in decreasing order based on the number of goals they have scored.	Order a list of stores in decreasing order based on their total yearly sales.

As seen in the table, the variations preserve the original idea and require similar skills to be completed, while the solution needed to complete each of them was different from the original version. To test if the variations were good enough, we asked students if they could identify which tasks were AI-generated. The total number of exercises in the seventh week was 21, meaning that there were 18 exercises generated by humans and 3 by ChatGPT. The exact wording of the question was "This week's tutorial featured a task or tasks generated by an AI tool. Did you recognize the task(s)?". The variations were created for the 2023 instance of the course, but we decided to include them in a similar form in 2024 as well to duplicate the amount of data.

The statistics of the student answers are provided in Table 2.

Table 2. Statistics about how well the students identified the AI-generated exercise variations from other, human-generated exercises. A similar data collection was performed in two instances (2023 and 2024) of the course.

Year	**2023**	**2024**
N	322	338
"I did not recognize"	91.30%	92.01%
Guessed wrong	5.90%	5.92%
Got 1 right	1.55%	0.59%
Got 2 right	0.93%	0.59%
Got all 3 right	0.00%	0.00%

The first answer "I did not recognize" means that the student reported not being able to distinguish the AI-generated exercises from the human-generated ones. The second, Guessed wrong, indicates that students thought they were able to identify an exercise (or several), but they were not the correct ones. The three final answers indicate the number of students who correctly identified 1, 2, or 3 AI-generated exercises, respectively. As seen in the table, the majority of students either said that they were not able to identify the exercise variations or tried to identify them, but got them wrong. Only a handful of students were able to recognize one or two exercises correctly, and no one was able to identify them all. We also checked the average score, used time, and number of submissions used for the three exercises, and found out that they are not different from the average values of all exercises in week 7.

4.4 Building a Web-Based Variation Tool

After the initial needs and feasibility of the concept had been confirmed in the prototype, the decision was made to transition the project into a deployable web page. The user interface of the web-based version is displayed in Fig. 2.

The tool supports importing exercises in multiple formats: the user can import individual exercises as a text file, but for a larger set, multiple exercises can be imported using an Excel sheet. In the case of importing the Excel file, users have two options. They can either process the questions one by one or they can export the whole Excel file containing variations for all exercises. This way modifying a collection of exercises, such as all exercises from a single week, can be done easily. An additional context can be added to the exercise, and the user can select a theme to be injected from a predefined list.

The generated variation of an exercise can contain a description, a skeleton (implementation to be filled by the student), and a model answer. Currently identifying these is implemented as a prompt. This means, that the tool is at the moment the most useful for generating programming exercises, and to enable the full support for other types of exercises may require some extra work and thorough testing.

5 Automated Test Generation

5.1 Background

As mentioned before, automated testing of programming assignments [3] is very important for providing students immediate feedback, which in turn enables them to fix and re-submit their solutions. As stated in various sources (see e.g. [5, 7, 28, 26]), the quality of feedback is very important as well. The feedback should provide enough information about the program submitted so that the student knows what sections of the program need more work. Typically, this means testing the individual functions of a modular program separately, a practice that is typically used in professional projects as well to test the quality of the programs.

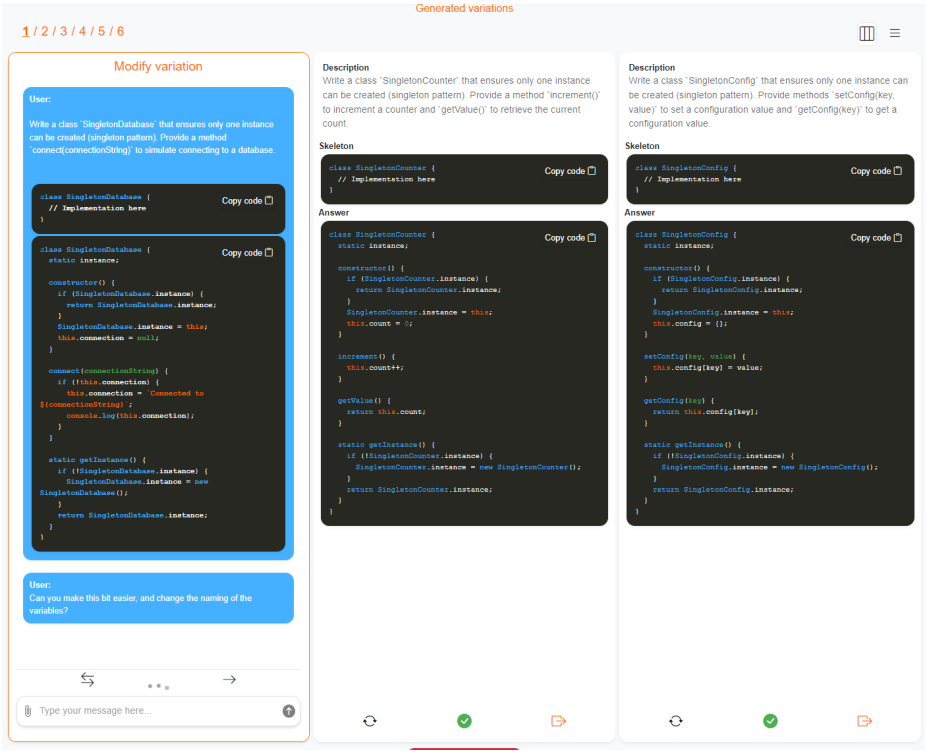

Fig. 2. The editing view of the web-based application displaying three variations generated. The user can ask for modifications to any variation by using the built-in, interactive chat window. The variations can be accepted or rejected, with rejected variations re-generated.

In the programming course, we previously provided feedback on the programs based on the outcome of the program. An example of this is displayed in Figure 3. In the example, the task was to write three functions, `sum`, `subtract`, and `multiply`, which all receive two parameters and return the value of the arithmetic operation. The task is one of the first tasks in the fourth week of the course, aimed at teaching the students function signatures and the syntax of the return statement, before moving on to more demanding tasks.

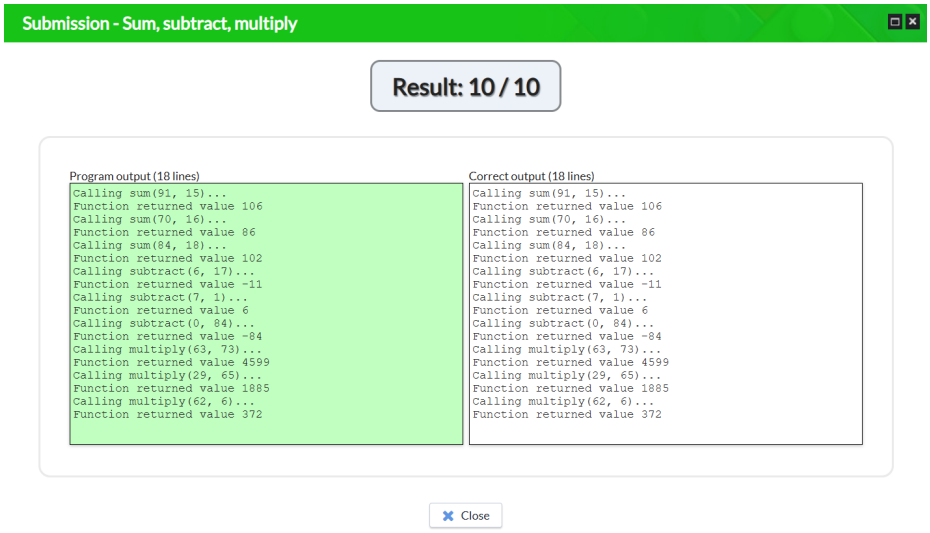

Fig. 3. An example feedback provided in the old system, displaying function calls and the return values.

5.2 Automated Unit Test Generation with ChatGPT

While the example feedback may be enough for simple exercises, providing enough information on more complex exercises requires a lot of work. Because of this, we wrote a testing and feedback template in Python. The template can test an individual function from various points of view, including for example the name of the function, the number and types of parameters (using Python's type hints [25]), the type of the return value, and the actual return values when called with different, pre-defined and/or randomly generated parameters.

An example of feedback provided by the testing template can be seen in Fig. 4.

We identified a total of 52 exercises in the course where the extended feedback could be useful. Modifying all these exercises one at a time to utilize the template would have been too difficult and a cumbersome task. Hence, we decided to

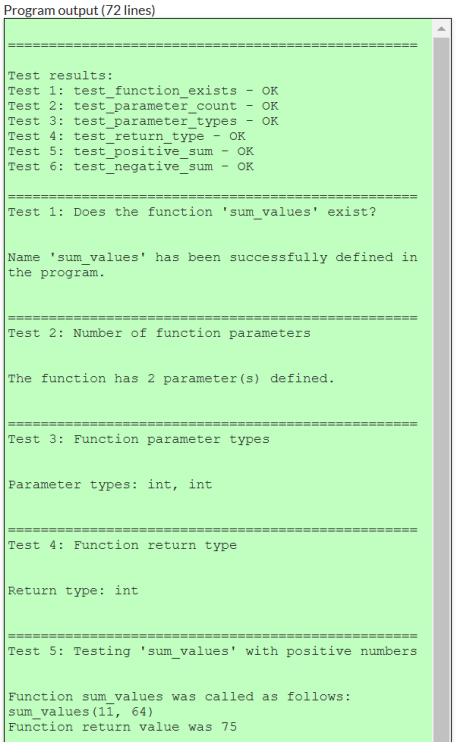

Fig. 4. An example feedback provided in the new system. The example feedback is truncated, the full feedback would show several different calls to the function with different parameter combinations and return values.

utilize ChatGPT for this. After several attempts, we came up with the optimized prompt to be used. The prompt structure is displayed in Figure 5.

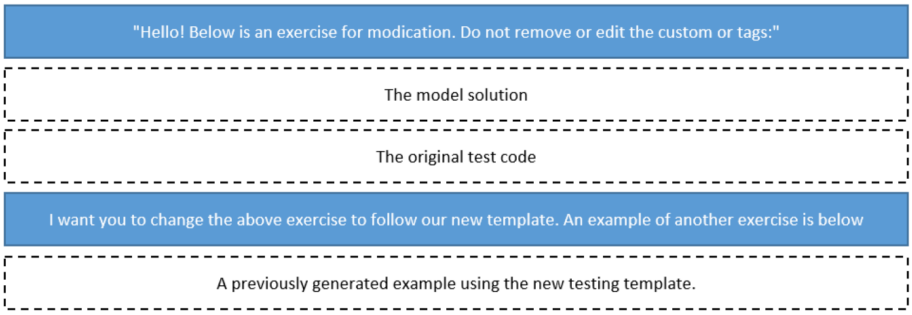

Fig. 5. The structure of the prompt used to generate the tests for an exercise. The text in solid blue frames is used as prompts, and the frames with dotted outlines present the resources included in the prompt.

The best outcome was achieved by providing ChatGPT with model solution and the original tests, and by including an example of previously generated program code utilizing the testing template. While testing the prompt, we realized that the 4o model of ChatGPT, which was the standardly used model when the paper was written, did not produce good enough results. Switching to (then) the latest model o1-preview provided results that could be utilized mostly as-is. Previously, the 4o model was deemed as the most suitable for programming-related tasks [27], but the o1 model was not yet available then. This underlines the quick development speed of models and the need for constant, ongoing research on their capabilities.

5.3 Results

To find out whether the new testing template adapted to exercises would provide different results, we compared two course instances to each other. The 2023 version of the course utilized the older version of the tests (Fig. 3) and the 2024 version the new tests (Fig. 4). In total, 52 exercises were included in the comparison. The results are displayed in Table 3.

As seen in Table 3, the mean score was a little lower in 2024, but the difference was quite small for modified and all exercises. It does however seem, that the modification lead to the lower amount of time spent on average. While there is a difference in time spent on all exercises as well, the change is more dramatic for the modified exercises. The mean number of submissions is also lower in 2024, but this time it seems that the change is smaller in modified exercises than in all exercises.

Table 3. Differences in the mean score, time on task, and number of submissions between 2023 and 2024 instances of the course. The test columns display the statistics for exercises that were modified to use the new testing template, while columns labeled "all" display the statistics for all exercises. Diff. columns display the relative change from 2023 to 2024.

	2023 test	2024 test	2023 all	2024 all	Diff. test	Diff. all
Exercises (N)	52	52	164	164		
Score	10.37	10.11	10.30	10.20	2.51%	0.96%
Time (s.)	5120.00	3075.67	3759.23	2967.27	39.93%	21.07%
Submissions	5.74	5.32	5.48	4.69	7.32%	14.40%

6 Discussion

The three exercise variations generated seemed to preserve the same characteristics as the original exercises. The students also seemed to perceive them similarly: almost all of them failed to distinguish them from the human-generated ones, and there was no difference in the statistics of completing them compared to other exercises. Hence, the answer to the first research question, "Is it possible to use an LLM to generate variations of the existing exercises with the same quality as the original ones?" is positive.

The reduction of nearly 40% of time used on the course exercises using the modified test framework is a notable improvement. It should be noted that according to [9], time usage and perceived difficulty level of the exercises seem to have only a small effect on the actual student performance on the course final exam performance. Therefore, the time reduction and score similarity displayed in the table 3 seems to support the enhanced student effectiveness. Hence, the answer to the second research question, "Is it possible to use an LLM to automatically create tests for introductory programming exercises?" is also positive.

It should be noted, that the performance increase in all submissions between 2023 and 2024 is probably at least partly explained by the addition of a *diff-checker*, which checks and highlights the differences in the student submission and the model answer. Another possible reason explaining the reduced time usage is the general popularization of AI tool usage. However, it should be noted that ChatGPT-3.5 was published as early as November 2022 [16], and its successor, GPT-4 on March 2023 [17]. As the 2023 instance of the course was held in the autumn of 2023, many of the tools available later in 2024 were available already then as well.

Even though the raw performance of the state-of-the-art LLM models increases, it is important to note that the need for specialized tooling has not disappeared. On the contrary, the benefits in the performance increase of LLMs in various tasks enable a fast development of specialized AI-driven tooling for specific use cases. As a more concrete example, high-quality models allow the tool development to focus on streamlining the work processes and reducing teachers' workload. This can be achieved more efficiently by focusing on usability aspects

instead of more technical aspects, such as the accumulation of suitable data, or a model fine-tuning.

Better models also pave the way to finding more suitable approaches for adaptive learning. When the reliability and the quality of the AI-generated exercises are high enough, in the future, it might be possible that they can be utilized directly by students or an autonomous agent controlling an adaptive learning path. This would enable personalized content [13] and truly adaptive exercises. However, the current performance of the models still has too many caveats to support this kind of approach.

7 Conclusion and Future Work

The rapid advancements of different AI tools are also visible in programming education. The current models enable reliable programming exercise generation (for specific use cases) and can reduce teachers' workload on updating course materials. The results of this study and the mentioned insights seem to validate the usability of models, at least in the specific tasks, in the context of programming education.

Future work should include exploring the limits of the current state-of-the-art models, especially on more complex programming exercise generation tasks. The scope of future work should also be broadened to cover other aspects of study material generation, such as updating out-of-date course materials. This is important, especially in the more technology-related fields such as computing, where the progress of technologies and tools is fast.

Finally, the effect of LLMs on programming education should be inspected more as a whole. LLMs change the working processes in industry, academia, and education alike, thus creating a need for re-assessing the scope and learning goals present in the current courses. Furthermore, the students are also becoming more accustomed to AI-driven approaches, which creates pressure for the teachers to stay up-to-date on the new advancements in the field of generative AI.

Acknowledgments. This work has been supported by FAST, the Finnish Software Engineering Doctoral Research Network, funded by the Ministry of Education and Culture, Finland.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Becker, B.A., Craig, M., Denny, P., Keuning, H., Kiesler, N., Leinonen, J., Luxton-Reilly, A., Prather, J., Quille, K.: Generative ai in introductory programming. Name of Journal (2023)
2. Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B.A., Reeves, B.N.: Prompt problems: A new programming exercise for the generative ai era. In: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. pp. 296–302 (2024)

3. Douce, C., Livingstone, D., Orwell, J.: Automatic test-based assessment of programming: A review. Journal on Educational Resources in Computing (JERIC) **5**(3), 4–es (2005)
4. Estévez-Ayres, I., Callejo, P., Hombrados-Herrera, M.Á., Alario-Hoyos, C., Delgado Kloos, C.: Evaluation of llm tools for feedback generation in a course on concurrent programming. International Journal of Artificial Intelligence in Education pp. 1–17 (2024)
5. Fuchs, M., Wolff, C.: Improving programming education through gameful, formative feedback. In: 2016 IEEE Global Engineering Education Conference (EDUCON). pp. 860–867. IEEE (2016)
6. Herden, O., et al.: Integration of chatbots for generating code into introductory programming courses. In: Conference Proceedings. The Future of Education 2024 (2024)
7. Jansen, J., Oprescu, A., Bruntink, M.: The impact of automated code quality feedback in programming education. In: Post-proceedings of the Tenth Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE). vol. 210 (2017)
8. Jury, B., Lorusso, A., Leinonen, J., Denny, P., Luxton-Reilly, A.: Evaluating llm-generated worked examples in an introductory programming course. In: Proceedings of the 26th Australasian Computing Education Conference. pp. 77–86 (2024)
9. Kaila, E., Rytilahti, J., Lokkila, E.: Utilizing learning analytics in large online courses. In: Education and New Developments 2024 - Volume II (2024)
10. Kazemitabaar, M., Ye, R., Wang, X., Henley, A.Z., Denny, P., Craig, M., Grossman, T.: Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In: Proceedings of the CHI Conference on Human Factors in Computing Systems. pp. 1–20 (2024)
11. Laakso, M.J., Kaila, E., Rajala, T.: Ville–collaborative education tool: Designing and utilizing an exercise-based learning environment. Education and Information Technologies **23**, 1655–1676 (2018)
12. Lekshmi-Narayanan, A.B., Oli, P., Chapagain, J., Hassany, M., Banjade, R., Brusilovsky, P., Rus, V.: Explaining code examples in introductory programming courses: Llm vs humans. arXiv preprint arXiv:2403.05538 (2023)
13. Logacheva, E., Hellas, A., Prather, J., Sarsa, S., Leinonen, J.: Evaluating contextually personalized programming exercises created with generative ai. In: Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1. pp. 95–113 (2024)
14. Lyu, W., Wang, Y., Chung, T., Sun, Y., Zhang, Y.: Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. In: Proceedings of the Eleventh ACM Conference on Learning@ Scale. pp. 63–74 (2024)
15. Meiβner, N., Speth, S., Becker, S.: Automated programming exercise generation in the era of large language models. In: 2024 36th International Conference on Software Engineering Education and Training (CSEE&T). pp. 1–5. IEEE (2024)
16. OpenAI: Chatgpt: Optimizing language models for dialogue. https://openai.com/index/chatgpt/ (nov 2022), accessed: February 13, 2025
17. OpenAI: Gpt-4 technical report. https://openai.com/research/gpt-4 (mar 2023), published March 14, 2023
18. Paiva, J.C., Leal, J.P., Figueira, Á.: Automated assessment in computer science education: A state-of-the-art review. ACM Transactions on Computing Education (TOCE) **22**(3), 1–40 (2022)

19. Paredes, C.M.G., Machuca, C., Claudio, Y.M.S.: Chatgpt api: Brief overview and integration in software development. International Journal of Engineering Insights **1**(1), 25–29 (2023)
20. Roest, L., Keuning, H., Jeuring, J.: Next-step hint generation for introductory programming using large language models. In: Proceedings of the 26th Australasian Computing Education Conference. pp. 144–153 (2024)
21. Rytilahti, J., Weerakoon, O., Kaila, E.: Exploring ai-driven programming exercise generation. In: Proceedings of the 20th International CDIO Conference, hosted by Ecole Supérieure Privée d'Ingénierie et de Technologies (ESPRIT) Tunis, Tunisia, June 10 – June 13, 2024 (2024)
22. Song, T., Zhang, H., Xiao, Y.: A high-quality generation approach for educational programming projects using llm. IEEE Transactions on Learning Technologies (2024)
23. Ta, N.B.D., Nguyen, H.G.P., Gottipati, S.: Exgen: Ready-to-use exercise generation in introductory programming courses. In: International Conference on Computers in Education (2023)
24. Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al-Ghamdi, A., Saleem, F.: The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. Computer Applications in Engineering Education **26**(6), 2328–2341 (2018)
25. Van Rossum, G., Lehtosalo, J., Langa, L.: Pep 484–type hints. Index of Python Enhancement Proposals (2014)
26. Venables, A., Haywood, L.: Programming students need instant feedback! In: Proceedings of the fifth Australasian conference on Computing education-Volume 20. pp. 267–272 (2003)
27. Wang, T., Zhou, N., Chen, Z.: Enhancing computer programming education with llms: A study on effective prompt engineering for python code generation. arXiv preprint arXiv:2407.05437 (2024)
28. Watson, C., Li, F.W., Lau, R.W.: Learning programming languages through corrective feedback and concept visualisation. In: Advances in Web-Based Learning-ICWL 2011: 10th International Conference, Hong Kong, China, December 8-10, 2011. Proceedings 10. pp. 11–20. Springer (2011)