USING LARGE LANGUAGE MODELS AS TOOLS IN INTRODUCTORY PROGRAMMING COURSES

Eli Roslöf

Department of Computer Science, Aalto University

Janne Roslöf

Faculty of Technology, LAB University of Applied Sciences

ABSTRACT

This paper explores the utilization of large language models (LLMs) in introductory programming courses in higher education. The primary objective is to examine how LLMs can support the creation of programming-related teaching materials. In addition, the aim is to reflect their potential impact on the learning objectives of these courses. The study is conducted as a literature review. The findings indicate that LLMs can be employed to generate a variety of programming tasks, such as small programming exercises and multiple-choice questions, with the quality of these tasks often comparable to those created by humans. Furthermore, LLMs enable the development of novel assignment types. In terms of learning objectives, the increasing prevalence of LLMs highlights the potential to introduce more advanced course content earlier. If LLMs are used to generate code, instructional emphasis may shift from code writing to reading, analyzing, and modifying code. Nevertheless, teaching fundamental programming concepts seems to remain essential, necessitating a balanced approach between leveraging LLMs and maintaining traditional programming practices.

KEYWORDS

Large language model, ChatGPT, Generative artificial intelligence, Programming education, Engineering education, Standards: 2, 8, 11

INTRODUCTION

Generative artificial intelligence (GAI) is transforming both working life and education. Many widely used tools, such as GitHub Copilot and OpenAI's ChatGPT, are freely available to students (OpenAI, n.d.), and their adoption among learners is increasing. Since its launch in November 2022, the number of active users of ChatGPT has continually increased from less than 100 million in 2023 to more than 700 million in 2025 (Chatterji et al., 2025). The development of AI tools has raised concerns among teaching staff. In a study conducted by Lau and Guo (2023), every interviewee mentioned the possibility of cheating, even though the interview questions did not explicitly address it. Many were also worried that students who rely heavily on AI might fail to understand course content. Consequently, several respondents emphasized the importance of supervised exams in assessment. Increasing the number of exams, however, was seen as only a short-term reaction to the growing use of AI tools.

Large language models also present opportunities. Lau and Guo (2023) identify capabilities that can be leveraged in teaching, such as code generation and explanation. AI could also assist in generating test cases and fixing errors. Modern LLMs are powerful enough to create programming exercises. Rytilahti, Weerakoon, and Kaila (2024) utilized ChatGPT to generate programming exercise descriptions using various strategies. While many exercises required slight manual adjustments, students were unable to distinguish between human-made and AI-generated versions of exercises. GAI can also be used to support students by creating personalized learning plans based on the students' needs (Chin & Ming, 2024). Utilizing generative AI can ease the workload for both students and instructors. However, a study conducted by Prather et al. (2025) found that fewer than half of programming course instructors include generative AI in their courses, but three-quarters believe that LLMs have changed the skills needed in software development. The most common reason for not adopting it was a lack of resources, be it skills or time, to do so.

This paper explores the utilization of LLMs in introductory programming courses in higher education. The primary objective is to examine how LLMs can support the creation of programming-related teaching materials. In addition, the aim is to study their potential impact on the learning objectives of these courses. This study is an extended version of the Bachelors' Thesis of the first author (Roslöf, 2024).

RESEARCH QUESTIONS AND METHODS

The goal of this study is to identify situations in which LLMs could be utilized for creating teaching materials for introductory programming courses. If the creation of various materials with the help of GAI is feasible, it could reduce the workload of teaching staff and enable a more personalized learning experience for students. In addition, this study aims to reflect the impact of LLMs on the learning objectives of these courses. LLMs may render many of the current course assignments trivial. Therefore, combining instructors' views with the opportunities offered by LLMs may provide insights into potential changes in course design. Thus, the following research questions have been defined for this work:

RQ1: In what types of programming-related teaching material creation for introductory programming courses can large language models be utilized?

RQ2: How is the increasing prevalence of large language models expected to affect the learning objectives of introductory programming courses?

RQ1 serves as the primary research question while RQ2 functions as a secondary, supporting question. Introductory programming courses refer to courses designed for first-year university students. Teaching materials, in this context, denote resources provided by course staff for student learning focusing on programming-related assignments.

Other teaching resources such as code examples, explanations describing code functionality or error messages, are excluded from this study. In addition, lecture videos or audio recordings, as well as support provided by course assistants, are excluded. The study also does not cover automated assessment of assignments or related feedback mechanisms. The discussion of RQ1 emphasizes positive opportunities and does not address ethical issues related to the use of GAI. RQ2 considers potential future changes. Furthermore, the study examines only changes related to individual courses, not broader institutional or degree program-level structures or policies.

This study was conducted as a literature review. Given the rapid development of LLMs, only references published after 2020 were included. The initial search for research articles was conducted in February 2024, and a supplementary search in December 2025. Consequently, this study reflects the state of LLM-related research as of December 2025. However, it should be noted that a large amount of research is being done on the topic. Google Scholar finds more than 3000 results published in 2025 matching the search terms "*Generative AI",* and *"computing education".* As such, the goal of this study is not to provide a systematic review of all existing literature. Instead, we aim at providing an overview into the field. To ensure access to the most recent findings, preprints from arXiv were also utilized, which may result in minor discrepancies. Most of the material consisted of peer-reviewed research articles and conference papers. Sources used to address RQ2 primarily comprised interview-based studies. Literature searches were conducted mainly through the Scopus and ACM databases using keywords such as *"programming education," "generative AI," "ChatGPT,"* and *"LLM evolution".* In addition to database searches, sources were identified through reference lists of other publications. Microsoft Copilot was utilized exclusively to assist with the initial translation of the original report (Roslöf, 2024) from Finnish to English. All other components of this study were conducted without the use of AI tools or software.

RESULTS

Modern tools that utilize GAI are the result of research spanning over a hundred years. The term "generative artificial intelligence" covers all computational techniques that enable the creation of material based on training data. The generated material can be almost anything, such as text or photographs. The content produced by GAI is meaningful and appears new—it does not simply reproduce the training data as-is. Generative AI is based on generative modeling, which is often combined with machine learning. This makes it possible to create new material based on learned information (Feuerriegel et al., 2024). Language models belong to the field of natural language processing (NLP) that combines computer science, AI, and linguistics. NLP is applied in tasks such as translation and text generation. Modern NLP uses deep learning to train language models (Li, 2022). Due to the rapid development of large language models, forming an up-to-date assessment of their capabilities is challenging. Prather et al. (2023) note that the abilities of the latest models may be underestimated in studies, and many commonly used performance evaluation methods are unsuitable when assessing LLMs from an educational perspective.

Programming-Related Assignments

A large part of the content in many introductory programming courses consists of various exercises. However, creating exercises is often a time-consuming process, and course staff may overlook aspects that are important for student learning. First, we discuss creating small programming exercises with the help of LLMs. After that, a new type of exercise where AI supports learning is introduced and, finally, the use of LLMs in creating multiple-choice questions that support the learning objectives is addressed.

Learner-Centered Exercises and Automatic Exercise Generation

Creating programming exercises is a laborious process, and considering the interests and skills of individual students in large courses is difficult. Students can also participate in creating exercises, but they seem not always to be very motivated to do so, and the quality of the questions produced varies. Small, automatically graded programming exercises are a common exercise type in introductory programming courses. Due to their small size, these exercises can help students practice specific skills, and completing multiple exercises promotes regular practice. In addition to the exercise description, a model solution and tests must be created. The difficulty level must also be carefully adjusted, as exercises that are too difficult or too easy affect student motivation and provide little learning benefit. Students in introductory courses often represent different skill levels and backgrounds. For this reason, Sarsa et al. (2022) point out that personalizing programming exercises based on student interests and skill levels can be beneficial for learning. Sharing solutions to programming exercises among students online is also unfortunately common.

According to Denny et al. (2022), attempts have been made to create practice exercises automatically using different approaches. Exercise creation often uses parameterized questions or question templates. However, they note that both generating questions based on templates and creating new, high-quality templates for different topics is labor-intensive. Thus, large numbers of programming exercises can be created by leveraging students. This results in a large exercise pool that can cover topics instructors might not include but that are important to students. However, learner-sourced exercises have two major problems in practice. First, even if students want to use exercises created by others, they are often not motivated to create them. Second, the quality of many student-created questions is generally low, limiting their usefulness for learning (Denny et al, 2022).

LLMs may be a key to creating large pools of personalized exercises. Sarsa et al. (2022) discuss this possibility and evaluate a set of exercises generated by OpenAI's Codex language model. The exercise set was created by providing the LLM with an example exercise that included the task description, a model solution, and automated tests. In addition, it was given a list of programming concepts and general themes, such as fishing, that should appear in the generated exercises. Only a small portion of the generated exercises were ready to use as-is. Sarsa et al. (2022) evaluated the generated exercises based on four criteria. The first criterion, meaningfulness, addresses whether the exercise description is reasonable and solvable by students. Three-quarters of the 120 evaluated exercises were classified as meaningful. The occurrence rate of the given programming concepts and themes ranged between 75% and 80%. The research team also assessed usability. Only 21% included both a model solution and automated tests that the solution passed. Of the exercises with both, one-third of the solutions passed the tests. Although the proportion of ready-to-use exercises was small, Sarsa et al. (2022) see potential in LLM-generated exercises. Modifying existing exercises and tests is easier than creating new ones, so even flawed outputs can be useful. Additionally, they suggested that students could create targeted and meaningful questions using keywords and

instructor-provided examples. Denny et al. (2022) also proposed a new type of exercise that uses LLMs to address problems in learner-sourced exercise creation. In robot-sourced exercise creation, the student creates an example exercise and keywords as described above. The LLM then generates multiple exercises based on the prompt.

A more complex exercise generation model is advantageous to creating high-quality exercises. Instead of using a single prompt, Jacobs et al. (2025) created a pipeline containing several prompts, each of which creating a part of the exercise. In addition to the exercise-generating prompts, it contained a reflection module with initiated up to five iterations of the generation process, if the created model solution could not pass the generated unit tests. As its input, the pipeline only required a context and a programming context which should be included in the exercise. They generated 200 exercises for different contexts, each requested to contain between one and three different introductory programming concepts. Based on expert evaluation, Jacobs et al. (2025) found that most exercises had correct syntax and were solvable based on the task description. All exercises incorporated the requested context. Out of the exercises deemed solvable, 80% had unit tests and 85% had a correct model solution. This is a significant improvement from the earlier work of Sarsa et al. (2022). However, the share of exercises that included the requested programming concepts decreased from 94% to 40% when the number of concepts was changed from one to three (Jacobs et al., 2025).

Jacobs et al. (2025) also demonstrated the potential of LLMs as a part of learner-sourcing. Using the exercise generation methodology described above, 26 introductory programming students participated in generating, solving, and evaluating exercises for which they could request contexts and concepts. The students rated the exercises similarly to the experts. Additionally, they found the process of generating personalized exercises interesting and useful for learning. They further recognized the usefulness of being able to generate and solve unlimited amounts of customized exercises (Jacobs et al., 2025). As students seem motivated to generate exercises and the generated exercises are mostly of high quality, the weaknesses of learner-sourcing noted by Denny et al. (2022) could be mitigated with LLMs.

Using LLMs for programming exercise creation is an attractive possibility that can increase the number of exercises available to students and reduce instructors' workload. Combining LLM use with learner-sourced exercise creation can address its problems and help students develop code evaluation skills.

Prompt Problems

The use of LLMs can also be taught and integrated into instruction. Prompt problems are a new exercise type developed by Denny et al. (2024), where students work with a LLM to solve a programming task. The exercise description is presented to the student visually rather than as a traditional text-based description so that the student cannot copy it directly into the LLM. Based on the description, the student develops a prompt, which is sent to the LLM. The LLM generates code, which is checked by an automated validator. The student sees both the code and the test results. If all tests are not passed, the student can generate a new prompt to try to get a better response from the LLM (Denny et al., 2024).

The main learning objective of prompt problems is teaching students how to create good code generation prompts. In a pilot study conducted by Denny et al. (2024), participants noted that prompt problems helped them practice computational thinking and problem-solving, as well as introduced them to previously unknown programming conventions. While most feedback was positive, some students were hesitant to use LLMs in programming. Additionally, prompt problems have been explored as a potential method for easing the learning process of non-

native English speakers. In a study by Prather, Reeves, et al. (2025), Arabic, Chinese, and Portuguese students solved prompt problems using their native languages. Most students were able to correctly solve the problems and found that using their native languages allowed for better expression of their thoughts. However, ChatGPT – the LLM used, had difficulties in understanding the participants' prompts, likely due to only a small number of its training materials being other than English. Programming languages and programming as a field being highly English-centric could also play a role (Prather, Reeves, et al., 2025). As the linguistic capabilities of LLMs improve, it's likely that the usefulness of prompt problems increases.

Although the previously mentioned studies had limited participation, prompt problems have been studied on a larger scale. Pădurean et al. (2025) expanded on the initial idea, adding more complex exercises, a chat interface allowing for iterative refinement of code, and making code execution available on request instead of happening automatically. Each of three different problem sets were attempted by over 900 students from an introductory programming course. In all cases more than 95% of students were able to complete the exercise. However, in two of the more complex problem sets, completion of the exercise did not require correct answers to all exercises. In addition to the high success rate, they found that students saw prompting to be easier than coding but did not seem overtly reliant on the LLM. Participants were more likely to request the execution of correct code rather than incorrect and thought that when solving difficult problems, a combination of prompting and code editing would be ideal (Pădurean et al., 2025).

Multiple-Choice Questions

There is a high demand for new, high-quality multiple-choice questions. This is influenced by growing interest in programming education and the sharing of answers to old exercises among students. Creating high-quality multiple-choice questions requires considerable expertise. If GAI could be used to create high-quality multiple-choice questions that support learning objectives, the workload of teaching staff would be significantly reduced (Doughty et al., 2024). Doughty et al. (2024) analyzed the ability of GPT-4 to create multiple-choice questions based on learning objectives and course descriptions: Approximately 250 learning objectives were collected from six different courses covering Python or data science fundamentals. As comparison material, 449 human-created multiple-choice questions were collected from these courses. Questions were classified according to the learning objective that were categorized into six levels based on the revised version of Bloom's taxonomy (Krathwohl, 2022). A system was created for generating multiple-choice questions that uses user input and static materials. Static materials include guidelines for creating multiple-choice questions, descriptions of Bloom's taxonomy levels and different question types, a small set of example questions, and instructions for returning results. Several questions were created for each learning objective—one for each question type defined for the objective's Bloom's taxonomy level.

The quality of the generated multiple-choice questions was comparable to human-created ones. Generated questions generally contained sufficient information, diverse options, and few errors in code. A larger proportion of these questions contained multiple correct options or revealed the correct answer in the question compared to human-created questions. However, the error rate was low, and compared to human-created questions, GPT-4 questions were better aligned with the learning objectives. Doughty et al. (2024) suggest that teaching staff focus more on aligning questions with the topic of the module rather than the learning objective. They also note that alignment of human-created questions with learning objectives was done manually, which may have influenced results (Doughty et al., 2024). The results

were promising and suggest that LLMs could be used to create better multiple-choice questions more quickly and with less effort.

Shu et al. (2025) expanded on the idea of utilizing learning objectives in multiple-choice question generation. Instead of learning objectives and course information, they used the name of the relevant module and a file containing the lecture notes as inputs. Additionally, the LLM received static materials similar to those of Doughty et al. (2024), but they did not include any example questions. Prior to generating the questions, the LLM was instructed to extract learning objectives aligning to Bloom's revised taxonomy from the lecture notes to use as a basis for the questions. They generated 66 questions based on four engineering courses' lecture notes, one of which was software engineering. Most of the questions were appropriate to the extracted learning objective and contained a well-constructed question stem. The answer option marked as correct was correct in 85% of cases, and in 72% of the questions the distractors were of high quality. Notably, all the cases with incorrect answers being marked as correct involved computations. Overall, Shu et al. (2025) noted that approximately 60% of the questions would be usable as-is, and approximately 30% could be modified to be usable. Additionally, they tested three of the generated questions on students, with the questions being mixed between human-generated questions. Approximately half of the students found the questions to be difficult, and the majority thought they were helpful in testing their understanding (Shu et al., 2025). As such, LLMs are capable of generating meaningful and non-trivial multiple-choice questions, although some human oversight is still needed.

Learning objectives

LLMs are likely to change what is taught in introductory programming courses. Research on this topic has described LLMs as a calculator-like tool that can reduce students' workload (Zastudil, 2023). The faculty opinions regarding the use of LLMs will influence what is taught in these courses in the future. If language models are utilized for code generation, introductory programming courses could cover topics traditionally reserved for advanced courses.

Denny et al. (2023) present several pedagogical approaches enabled by LLMs. If students do not need to focus on learning syntax, algorithmic concepts could be introduced earlier. They also suggest that assignments could shift from implementation to specification creation. Tasks might focus on designing prompts for LLMs rather than writing code. Moreover, language models can generate code that helps students start an assignment or project, targeting the focus on code modification, refactoring, and debugging skills. Emphasizing testing and performing it multiple times during an assignment could also help students understand both their own code and the code generated by LLMs (Denny et al., 2023).

The attitudes of programming course instructors toward changes brought by LLMs have been studied in several papers. Prather et al. (2023) interviewed 22 course instructors about LLMs' impact on their teaching. One topic discussed was the effect of LLMs on course learning objectives. Lau & Guo (2023) interviewed 20 members of introductory programming course staff, asking them to consider changes they would make if students had access to a perfect, invisible AI tool capable of generating and explaining code. Although the description was hypothetical, they note that many interviewees referred to existing tools such as ChatGPT. Both studies revealed concerns about student learning when using LLMs in introductory courses. The interviewees emphasized that understanding fundamental programming principles remains essential, even if language models perform part of the work (Lau & Guo, 2023; Prather et al., 2023). Some respondents did not want to change the learning objectives, as these topics form the foundation of programming knowledge (Prather et al., 2023). Lau & Guo (2023) report that some instructors critical of LLMs aimed to restrict their use, for example,

by adding more paper-based exams. Their interviewees noted that leveraging LLMs would allow complex topics, such as software design, to be introduced earlier. Courses could focus more on reading, modifying, and critically analyzing code rather than writing it.

Some instructors have already altered their courses as a result of the emergence of LLMs. As a part of a larger study, Prather, Leinonen, et al. (2025) conducted a survey on 76 persons involved in computing education, most of whom teach at universities, and interviewed 12 educators using or researching generative AI. More than three quarters of the respondents felt that the skills needed in software development have changed due to LLMs. However, only a third of them had included GAI in their courses. A minority had taken action to prevent LLM usage, with techniques focusing on examining student-written code and changing existing assignments. The respondents who had changed their courses reported focusing more on reading and editing code as well using LLMs effectively. This higher level of abstraction was deemed to be more important than learning syntax, which echoes the ideas presented by Denny et al. (2023). Additionally, Prather, Leinonen, et al. (2025) found that instructors had increased proctoring of assignments and exams.

Across the studies reviewed, two contrasting perspectives emerged. On one hand, LLMs could enable earlier coverage of complex topics, such as algorithms, in introductory courses. Many also believe teaching should increasingly emphasize code analysis and modification. On the other hand, some argue that core programming concepts must still be taught in introductory courses, so learning objectives should remain unchanged. Among educators from other disciplines and educational levels, there was interest in teaching LLM usage, and only a small minority would ban them entirely. Most studies addressed anticipated future changes, as educators have not yet fully adapted to the rapid development of LLMs. The changes that have already been made are quite similar to the expected changes. Actions are being taken to both include LLMs into learning objectives and ensure assessment results reflect student skills instead of the answering capabilities of LLMs. Institutional policies, such as those of universities, may also influence how LLMs are incorporated into course objectives.

DISCUSSION

This study has provided an overview of several types of programming-related teaching materials generated by LLMs that could be utilized in programming courses. The creation of these materials supports the presented learning objectives. For example, the prompt-based tasks help students develop skills in reading and analyzing code while also teaching effective use of language models. However, it should be noted that the practical use of LLMs for creating teaching materials is still developing rapidly, making it difficult to assess their future potential or the extent to which their adoption will become widespread. Furthermore, many material types—such as prompt problems– have only been examined by limited number of researchers. Future research should therefore incorporate data from multiple research teams. It is also important to recognize that the quality of LLM outputs depends heavily on the prompts; if they are incomplete, the results may be worse than the model's actual capabilities.

LLMs could also enable the creation of content types not addressed in this study. For example, generating general text-based course materials based on the learning objectives could be possible. LLMs could also be used for more complex tasks, such as creating specifications for programming projects. Furthermore, many introductory programming courses include tasks where students complete partially written code. LLMs could generate these code fragments. They could also create UML diagrams that students would use for implementing programs. LLMs could also enable the creation of interactive tasks like prompt problems. For instance,

coding style practices could be taught by having students submit their code to an LLM, which would provide feedback on aspects such as variable naming.

Learner-generated tasks have traditionally suffered from low student motivation. Assistance from LLMs could enable the creation of larger pools of tasks and help improve motivation. Learner-centered approaches could also be applied to refining LLM-generated materials through identifying mistakes in them. Assignments focused on error correction would help instructors produce high-quality AI-based materials with minimal effort while training students to identify and fix mistakes. Similarly, refactoring could be taught more effectively if prompts instruct the LLM to generate poorly structured code. The ability of LLMs to personalize materials also allows for more individualized and engaging learning experiences. Personalization could also account for differences in students' backgrounds and provide varying levels of support based on individual needs. However, the accessibility of LLMs means that students must take greater responsibility for their own learning. They can use these tools to solve a large portion of assignments, often without instructors noticing. Therefore, students themselves bear responsibility for how they use LLMs. If a student relies on an LLM to produce all answers without engaging with the material, meaningful learning is unlikely to occur. Students should also recognize that LLMs can be used productively in learning—for example, to generate new tasks at different difficulty levels, or explain programming concepts repeatedly until they are understood.

Although not within the scope of this paper, the use of LLMs in automated assessment should be considered. Nagakalyani et al. (2025) found that an LLM-based grading assistant could speed up the process of manually grading assignments in an introductory programming course by almost a quarter on average. While LLMs can create feedback for programming exercise, Er et al. (2025) found that human-generated feedback was perceived as more useful by students, and its usage led to greater improvements in assignment scores compared to LLM-generated feedback. Regardless of the technical capabilities of LLMs in grading or creating feedback for assignments, human oversight is needed. Furthermore, the ethical implications of using LLMs in assesses should be considered before implementation.

CONCLUSIONS

This study aimed to identify ways in which LLMs can be utilized in creating teaching materials for introductory programming courses. Based on the findings, LLMs can be applied to the generation of both programming tasks and multiple-choice questions. The created materials are of high quality, and a portion is ready for use in teaching. LLMs can also be used to create prompt-based exercises, where the generated code helps students practice writing prompts and analyzing code. Language models can further support the creation of other types of materials not discussed in this study. For example, LLMs can produce entirely new code examples and explanations (Sarsa et al., 2022; MacNeil et al., 2023; Jury et al., 2024). In addition, LLMs can be used for explaining error messages that are typically challenging for many students (Leinonen et al., 2023).

It was also explored how LLMs may influence learning objectives in programming courses. With the help of language models, it could be possible to incorporate topics such as algorithms and software design into introductory courses. Teaching could also place greater emphasis on code analysis and modification rather than solely on code writing. Nevertheless, teaching the fundamentals of programming is still considered essential, and not all stakeholders view changes to learning objectives as justified.

The rapid advancement of LLMs and other AI tools is likely to have significant practical implications for the development of engineering education in general, including the CDIO framework and its associated practices. For example, although the current version of the CDIO Syllabus (Malmqvist et al., 2022) identifies digitalization as a key enabling technology that supports engineers in addressing emerging and existing challenges more effectively, the accelerating evolution of AI tools—and their impact on the competencies expected of graduating engineers—will need to be addressed in future revisions.

This study could be expanded in several ways, such as by including empirical studies, evaluating practical classroom implementations, or exploring ethical and pedagogical implications of LLM use in programming education. Beyond learning objectives, future research could also explore the potential of LLMs to act as course assistants.

ACKNOWLEDGEMENTS

The kind guidance of Professor Lauri Malmi (Aalto University, Finland) is gratefully acknowledged.

The authors received no financial support for this work.

REFERENCES

Chatterji, A., Cunningham, T., Deming, D. J., Hitzig, Z., Ong, C., Shan, C. Y., & Wadman, K. (2025) *How People Use ChatGPT*. NBER Working Paper 34255 https://doi.org/10.3386/w34255

Chin, C. S., & Ming, K. W. (2024) Boosting Performance with AI-Powered Adaptive Learning. *Proceedings of the 20th International CDIO Conference,* pp. 620-630, Tunis, Tunisia.

Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B. A., & Reeves, B. N. (2024). Prompt Problems: A New Programming Exercise for the Generative AI Era. *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education,* pp. 296-302. https://doi.org/10.1145/3626252.3630909

Denny, P., Prather, J., Becker, B. A., Finnie-Ansley, J., Hellas, A., Leinonen, J., Luxton-Reilly, A., Reeves, B. N., Santos, E. A., & Sarsa, S. (2023). Computing Education in the Era of Generative AI. *arXiv*. https://doi.org/10.48550/arXiv.2306.02608

Denny, P., Sarsa, S., Hellas, A., & Leinonen, J. (2022), Robosourcing Educational Resources - Leveraging Large Language Models for Learnersourcing. *Proceedings of the Workshop on Learnersourcing: Student-Generated Content @ Scale 2022*, Vol. 3410, New York City, New York, USA. https://ceur-ws.org/Vol-3410/paper1.pdf

Doughty, J., Wan, Z., Bompelli, A., Qayum, J., Wang, T., Zhang, J., Zheng, Y., Doyle, A., Sridhar, P., Agarwal, A. S., Bogart, C., Keylor, E., Kultur, C., Savelka, J., & Sakr, M. (2024). A Comparative Study of AI-Generated (GPT-4) and Human-crafted MCQs in Programming Education. In Herbert, N., & Seton, C. (eds.) *Proceedings of the 26th Australasian Computing Education Conference,* pp. 114-123, Sydney, Australia. https://doi.org/10.1145/3636243.3636256

Er, E., Akçapınar, G., Bayazıt, A., Noroozi, O., & Banihashem, S. K. (2025). Assessing student perceptions and use of instructor versus AI-generated feedback. *British Journal of Educational Technology*, 56(3), pp. 1074–1091. https://doi.org/10.1111/bjet.13558

Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2024). Generative AI. *Business & Information Systems Engineering*, 66(1), pp. 111-126. https://doi.org/10.1007/s12599-023-00834-7

Jacobs, S., Peters, H., Jaschke, S., & Kiesler, N. (2025). Unlimited Practice Opportunities: Automated Generation of Comprehensive, Personalized Programming Tasks. *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education*, pp. 319–325, New York, NY, USA. https://doi.org/10.1145/3724363.3729089

Jury, B., Lorusso, A., Leinonen, J., Denny, P., & Luxton-Reilly, A. (2024). Evaluating LLM-generated Worked Examples in an Introductory Programming Course. *ACE 2024 - Proceedings of the 26th Australasian Computing Education Conference*, pp. 77-86, Sydney, Australia. https://doi.org/10.1145/3636243.3636252

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory Into Practice*, 41(4), pp. 212–218. https://doi.org/10.1207/s15430421tip4104_2

Lau, S., & Guo, P. (2023). From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. *Proceedings of the 2023 ACM Conference on International Computing Education Research, Volume 1*, pp. 106-12, Chicago, IL, USA. https://doi.org/10.1145/3568813.3600138

Leinonen, J., Hellas, A., Sarsa, S., Reeves, B., Denny, P., Prather, J., & Becker, B.A. (2023). Using Large Language Models to Enhance Programming Error Messages. *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, pp. 563–569, Toronto, Canada. https://doi.org/10.1145/3545945.3569770

Li, H. (2022). Language models: past, present, and future. Commun. ACM 65, 7 (July 2022), pp. 56–63. https://doi.org/10.1145/3490443

MacNeil, S., Tran, A., Hellas, A., Kim, J., Sarsa, S., Denny, P., Bernstein, S., & Leinonen, J. (2023). Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, pp. 931-937, Toronto, Canada. https://doi.org/10.1145/3545945.3569785

Malmqvist, J., Lundqvist, U., Rosén, A., Edström, K., Gupta, R., Leong, H., Cheah, S-M., Bennedsen, J., Hugo, R., Kamp, A., Leifler, O., Gunnarsson, S., Roslöf, J., & Spooner, D. (2022). The CDIO Syllabus 3.0 - An Updated Statement of Goals. *Proceedings of the 18th International CDIO Conference*, pp. 18-36, Reykjavik, Iceland.

Nagakalyani, G., Chaudhary, S., Apte, V., Ramakrishnan, G., & Tamilselvam, S. (2025). Design and Evaluation of an AI-Assisted Grading Tool for Introductory Programming Assignments: An Experience Report. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSETS 2025).* pp. 805–811. https://doi.org/10.1145/3641554.3701913

OpenAI (n.d.). *Pricing*. Retrieved December 20, 2025, from https://chatgpt.com/pricing/

Pădurean, V.-A., Denny, P., Gotovos, A., & Singla, A. (2025). Prompt Programming: A Platform for Dialogue-based Computational Problem Solving with Generative AI Models. *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education*, pp. 458-464, New York, NY, USA. https://doi.org/10.1145/3724363.3729094

Prather, J., Denny, P., Leinonen, J., Becker, B. A., Albluwi, I., Craig, M., Keuning, H., Kiesler, N., Kohn, T., Luxton-Reilly, A., MacNeil, S., Peterson, A., Pettit, R., Reeves, B. N., & Savelka, J. (2023). The Robots are Here: Navigating the Generative AI Revolution in Computing Education. *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*, pp. 108-159, Turku, Finland: ACM. https://doi.org/10.1145/3623762.3633499

Prather, J., Leinonen, J., Kiesler, N., Gorson Benario, J., Lau, S., MacNeil, S., Norouzi, N., Opel, S., Pettit, V., Porter, L., Reeves, B. N., Savelka, J., Smith, D. H. IV, Strickroth, S., & Zingaro, D. (2025). Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. *2024 Working Group Reports on Innovation and Technology in Computer Science Education,* pp. 300-338, New York, NY, USA: ACM. https://doi.org/10.1145/3689187.3709614

Prather, J., Reeves, B. N., Denny, P., Leinonen, J., MacNeil, S., Luxton-Reilly, A., Orvalho, J., Alipour, A., Alfageeh, A., Amarouche, A., Kimmel, B., Wright, J., Blake, M., & Barbre, G. (2025) Breaking the Programming Language Barrier: Multilingual Prompting to Empower Non-Native English Learners. *Proceedings of the 27th Australasian Computing Education Conference,* pp. 74-84, New York, NY, USA: ACM. https://doi.org/10.1145/3716640.3716649

Roslöf, E. (2024). *Large Language Model as a Tool in Introductory Programming Courses*. B.Sc. (Tech.) Thesis report, Aalto University, Finland [in Finnish]. Available at: https://urn.fi/URN:NBN:fi:aalto-202405283867

Rytilahti, J., Weerakoon, O., & Kaila, E. (2024). Exploring AI-driven Programming Exercise Generation. *Proceedings of the 20th International CDIO Conference*, pp. 708-716, Tunis, Tunisia.

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, pp. 27-43, Lugano, Switzerland. https://doi.org/10.1145/3501385.3543957

Shu, C., Yao, N., Chen, Y., Wijeratne, V., Ma, L., Loo, J., Chai, K. K., Alam, A. Abuelmaatti, A. (2025) AI-Assisted Multiple-Choice Questions Generation With Multimodal Large Language Models in Engineering Higher Education *2025 IEEE Global Engineering Education Conference,* London, United Kingdom. https://doi.org/10.1109/EDUCON62633.2025.11016449

Zastudil, C., Rogalska, M., Kapp, C., Vaughn, J., & MacNeil S. (2023). Generative AI in Computing Education: Perspectives of Students and Instructors *2023 IEEE Frontiers in Education Conference*, College Station, TX, USA. https://doi.org/10.1109/FIE58773.2023.10343467

BIOGRAPHICAL INFORMATION

Eli Roslöf is currently enrolled in the Master's Degree Programme in Computer, Communication and Information Science with a major in Computer Science at Aalto University, Finland. Currently, they are completing a master's thesis at Nokia Solutions and Networks. Prior to this industrial role, they served as a teaching assistant for several introductory programming courses. Their research interests include applied machine learning and artificial intelligence as well as introductory programming education.

Janne Roslöf serves as the Dean of the Faculty of Technology at LAB University of Applied Sciences, Finland. He holds a D.Sc. and M.Sc. in Process Systems Engineering from Åbo Akademi University (ÅAU) (Finland) as well as an M.A. in Education Science from the University of Turku (Finland). Also, he is an Adjunct Professor of Software Engineering Education at the Faculty of Science and Engineering of ÅAU. His research interests include project-based learning environments, first-year engineering student experience as well as topics connected to student-flow in higher engineering education.

Corresponding author

Eli Roslöf
Department of Computer Science
Aalto University
Konemiehentie 2
02150 Espoo, FINLAND
eli.roslof@aalto.fi