Evaluating Contextually Personalized Programming Exercises Created with Generative AI

Evanfiya Logacheva
Aalto University
Espoo, Finland
evanfiya.logacheva@aalto.fi

Arto Hellas
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

James Prather
Abilene Christian University
Abilene, TX, USA
james.prather@acu.edu

Sami Sarsa
University of Jyväskylä
Jyväskylä, Finland
sami.j.sarsa@jyu.fi

Juho Leinonen
Aalto University
Espoo, Finland
juho.2.leinonen@aalto.fi

ABSTRACT

Programming skills are typically developed through completing various hands-on exercises. Such programming problems can be contextualized to students' interests and cultural backgrounds. Prior research in educational psychology has demonstrated that context personalization of exercises stimulates learners' situational interests and positively affects their engagement. However, creating a varied and comprehensive set of programming exercises for students to practice on is a time-consuming and laborious task for computer science educators. Previous studies have shown that large language models can generate conceptually and contextually relevant programming exercises. Thus, they offer a possibility to automatically produce personalized programming problems to fit students' interests and needs. This article reports on a user study conducted in an elective introductory programming course that included contextually personalized programming exercises created with GPT-4. The quality of the exercises was evaluated by both the students and the authors. Additionally, this work investigated student attitudes towards the created exercises and their engagement with the system. The results demonstrate that the quality of exercises generated with GPT-4 was generally high. What is more, the course participants found them engaging and useful. This suggests that AI-generated programming problems can be a worthwhile addition to introductory programming courses, as they provide students with a practically unlimited pool of practice material tailored to their personal interests and educational needs.

CCS CONCEPTS

• **Social and professional topics → Computing education**; • **Human-centered computing → Human computer interaction (HCI)**; • **Computing methodologies → Artificial intelligence**.

ICER '24 Vol. 1, August 13–15, 2024, Melbourne, VIC, Australia
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0475-8/24/08
https://doi.org/10.1145/3632620.3671103

KEYWORDS

generative AI, large language models, automatic exercise generation, context personalization

ACM Reference Format:
Evanfiya Logacheva, Arto Hellas, James Prather, Sami Sarsa, and Juho Leinonen. 2024. Evaluating Contextually Personalized Programming Exercises Created with Generative AI. In *ACM Conference on International Computing Education Research V.1 (ICER '24 Vol. 1), August 13–15, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3632620.3671103

1 INTRODUCTION

Learning how to program involves developing various sets of programming skills. Computing education theories present them as distinct and often sequential or hierarchical, i.e., meant to be introduced in a certain order [21, 52, 90]. Since developing expertise in any domain requires a certain amount of training [29], developing programming skills relies on practical hands-on exercises focused on particular learning objectives [2, 15, 50, 55, 68, 77, 82, 90].

Research indicates that many students in computer education experience high levels of stress, frustration, confusion, and boredom, which can lead to negative outcomes [11, 82]. Frustration is often developed as a result of encountering various difficulties related to both coding itself and course instruction; it precedes boredom and loss of interest in subject learning [11]. Certain solutions such as substituting long weekly assignments with multiple shorter problems have been reported to alleviate students' stress and improve their performance [2]. Similarly, repetitive exercises designed for teaching students syntax in CS1 have been found effective for enhancing students' engagement and their exam scores [15]. Such exercises have been viewed as helpful by students [50, 77]. Overall, learning how to program hands-on has been reported to reduce students' stress levels [82].

Programming exercises can be contextualized, i.e., worded in a narrative, or decontextualized, when they are devoid of context [6, 12, 13, 26, 44, 48, 49]. Providing context is thought to improve students' engagement by making course materials relevant [26]. Research shows mixed evidence for using contextualization in computing education. Studies using the Rainfall Problem and the Satellite Problem have found little effect on students' performance when it comes to solving contextualized vs. decontextualized programming problems [6, 12, 48, 49]. However, Leinonen et al. [44] have discovered that context might help students avoid algebraic errors when they are tasked with programming exercises that involve mathematics. Guzdial [25] has reported an increase in course participant retention when contextualized exercises were introduced in course materials. In the field of educational psychology, there has been a number of studies on context personalization of mathematical problems [5, 34, 35, 72, 86]. Context personalization means tailoring learning materials to personal interests, preferences, and cultural backgrounds of learners [5, 34, 35, 72, 73, 85–87]. The contextualization approach centered around students' personal interests or preferences has demonstrated positive results on situational interest and perceived utility value among students with low interest in mathematics [34, 35]. It has also positively impacted both students whose quantitative engagement with their interests was high, in case of deep contextualization, and low, when context involved only superficial details [86]. In a longitudinal study by Bernacki and Walkington [5], context personalization has improved students' interest in mathematics and their test results. There has been no similar research on context personalization in computing education, as studies on contextualization in the domain of computer science have investigated whether it has any utility [6, 12, 44, 48, 49]. Considering possible positive impact of contextualization on students' motivation and engagement [26], Bouvier et al. [6] note that investing time and effort into contextualizing programming problems might be worthwhile. As they point out, developing and assessing such problems is a labor-intensive endeavor for educators, which is why there has been a shift towards researching automatic generation of programming exercises [13, 70].

Recent research on large language models (LLMs) has demonstrated promising applications of artificial intelligence in computing education [3, 7, 19, 20, 37, 40, 41, 43, 45, 47, 51, 62, 89]. LLMs can generate code explanations [43, 51] and programming feedback [32, 38, 40, 41, 56], make programming error messages more comprehensible [45, 69, 88], enhance intelligent tutoring systems [7] and coding assistants [47], solve programming problems [19, 20, 63, 89], generate worked examples [31, 37] and programming exercises [13, 36, 70]. Sarsa et al. [70] have shown that LLMs can be successfully used for automatically generating new exercises, containing a problem description, a sample solution, and test code. In their study, LLMs could create thematically and conceptually relevant exercises. A subsequent study by del Carpio Gutierrez et al. [13] has also demonstrated that LLMs are capable of creating high-quality exercises that include various contextual narratives in their problem descriptions and working code solutions.

Previous research on generating novel programming problems with personalized context narratives and context personalization have inspired this work dedicated to automatic generation of programming exercises [4, 5, 13, 34, 35, 70, 72, 85–87]. LLMs offer a possibility to provide students with personalized exercises for hands-on practice that could alleviate existing difficulties students face when learning programming. In this work, we study the quality of programming exercises generated by LLMs, specifically GPT-4. We have both the study authors and students review the exercises. We also examine how they interact with the LLM-generated exercises in an elective introductory programming course offered by Aalto University and their feedback on the contextual personalization of the exercises.

We seek to answer the following questions:

- RQ1. How do the study authors evaluate the quality of contextually personalized exercises generated by GPT-4?
- RQ2. How do students evaluate the quality of contextually personalized exercises generated by GPT-4?
- RQ3. How do students interact with the tool that provides contextually personalized exercises?

The contributions of our work are the following:

- We examine the quality of the LLM-generated programming exercises, finding that the study authors and the course participants rate them highly.
- We report that the course participants give overwhelmingly positive feedback on the LLM-generated programming exercises, suggesting that using them as supplementary exercises is well received by students.
- We study the interactions the course participants have with the generated exercises, finding that they prefer to choose the theme of the exercises over receiving a random exercise, which suggests that contextual personalization of exercises with LLMs could be an effective way to increase student engagement.

2 RELATED WORK

2.1 Context Personalization

Personalization in education has various applications and definitions. According to Bernacki et al. [4], the most common features mentioned in its definitions are related to identifying and adapting instruction to the needs and interests of learners. However, these terms vary across educational approaches to personalization. Tetzlaff et al. [78] contrast personalized education with traditional approaches that disregard individual characteristics of students and treat them as a unified group.

In addition to systems that work with individualized learning needs of students, there are context personalization strategies offering learning materials according to students' out of school interests, preferences, or cultural backgrounds [5, 34, 35, 72, 73, 85–87]. Solari et al. [73] summarize what constitutes learners' interests in educational psychology literature. According to them, the majority of research considers a particular type of relationship between a person and their interest, their psychological experiences and focuses on the engagement involved in a personal interest. However, they note the existing diversity of theoretical approaches to defining the object of an interest, centered around topics, domains, and practices. Solari et al. [73] propose that the personalization strategy that works with students' individual interests enhances their sense of value and meaning associated with learning. Additionally, context personalization can engage contextual grounding, which utilizes students' prior experiences with a particular context [5]. It is thought to ease long-term memory recollection, lessen the probability of conceptual errors, and facilitate understanding of subject domain concepts [5].

It has also been suggested that context personalization may stimulate students' situational interest, which is characterized by improved engagement and attention caused by conspicuous parts of learners' environments [5, 33–35, 54, 86]. Situational interest, in turn, can lead to the development of individual interest in a subject domain, for example, mathematics or computing [5, 33, 34, 54]. Hidi and Renninger [33] suggest a four-phase model of situational and individual interest development. The first phase is triggered situational interest, which may serve as a precursor to the inclination to engage with particular content over a period of time; the second phase, maintained situational interest, is characterized by focused attention and may nudge learners towards further engagement in a more advanced phase. Individual interest, comprised of emerging and well-developed (well-established) phases, is typically self-generated, while situational interest is often externally supported [33]. Nevertheless, emerging individual interest might be hindered by lack of support, resources, and positive reinforcement, whereas well-established interest is stable throughout time [54]. In computing education, developing individual interest is thought to rely on consistent positive situational interest built over time, along with discovering value and knowledge in educational content and building "a sense of belonging" to it [54]. Thus, contextually personalized learning materials could trigger the initial stage of situational interest development and help learners develop a sense of attachment and value in educational content.

Context personalization approaches differ in terms of the depth of implementation. Surface personalization involves simple references to students' interests, while deep personalization draws on students' prior knowledge and uses it in learning materials [85–87]. Personalization also diverges at the level of granularity [87]. At a fine grain size, personalized problems focus on particular topics, for example, a student's favorite ice hockey teams. In contrast, when they center around domains, e.g., sports, personalization is applied at a broader level of granularity. Additionally, context personalization strategies can be based on either interests or preferences [35]. Here, preferences are options from students' mundane life aspects, for instance, friendships and food, while interests pertain to objects of sustained engagement, e.g., music and sports [35]. Context personalization is also used as an adaption of problems to real-life contexts experienced by students without involving their personal interests. Schoenherr [72] used locations and objects of students' hometown for personalizing problems.

While a series of empirical studies on context personalization has been conducted on children in mathematics classes or after-school programs [5, 34, 35, 72, 86], there has been little attention dedicated to it in computing education. Høgheim and Reber [34] found that in mathematics, surface context personalization positively affected situational interest and perceived utility value among students whose subject area interest was low. Their subsequent study showed that personalization based on individual preferences mostly benefited learners with low interest and perceived competence in mathematics [35]. The authors pointed out that the level of granularity of personalization could have an effect on situational interest. Similarly, Walkington and Bernacki [86] discovered that the depth of personalization influenced its effectiveness. Learners who were less involved with quantitative functions in their interests were more likely to be positively affected by surface personalization, while those whose quantitative engagement with their interests was higher benefited more from deep personalization. Bernacki and Walkington [5] reported an improvement in developing students' interest in mathematics through context personalization in a longitudinal study. They also noted a beneficial influence of solving personalized problems on students' test performance. In a study conducted as an after-school mathematics program, using familiar locations in context personalization was found to increase students' intrinsic (interest and enjoyment) and attainment (related to personal or identity-based importance) values [14, 72].

Contextually personalized problems have a potential to solve a number of issues associated with frustration and boredom experienced by students in programming education [11]. They are, nevertheless, time-consuming to create [6]. However, as LLMs display state-of-the-art results in many natural language processing problems and even code generation [9, 19, 20, 40, 41, 46, 63, 80, 89], creating contextually personalized programming exercises might be achieved with their help.

2.2 Perceptions of Assessment Quality

The way students and teachers perceive assessment is relevant to this study as we generate uncredited, additional practice opportunities for students for formative assessment utilizing generative AI. Struyven et al. [76] conducted a literature review of 36 empirical studies. They found that, in general, students find assessment positive (i.e., beneficial for their learning) and fair (i.e., accurately and justly measuring their progress towards their learning goals) if the assessment relates to authentic tasks, presents reasonable demands, encourages them to apply knowledge to realistic contexts, focuses on the need to develop a range of skills, and is perceived to have long-term benefits [67, 76].

This is supported by later empirical findings. Van Dinther et al. [81] studied the perceptions of assessment of 138 first-year elementary teacher students at a Dutch university. They focused on studying the links between perceptions, self-efficacy, and performance. They found that formative assessment where students create "a quality product or observable performance in a real-life situation" and where feedback is tied to the task and criteria increases self-efficacy, which in turn is likely to lead to more learning. They also argue that the presence of sufficient practice is a requirement for mastery of the topic.

Gerritsen et al. [22] studied the perceptions of 204 higher education students based on six aspects of assessment quality: effects on learning, fairness, conditions, interpretation of test scores, authenticity, and credibility. They found that students who had more positive perceptions of the effects of assessment on learning were more likely to employ deeper and strategic learning approaches, whereas students who had negative perceptions were more likely to apply a surface learning approach, which has been linked to worse learning outcomes [53]. They argue that this is due to students deepening their approach if they find the assessment appropriately challenging and motivating. Similarly, Gulikers et al. [24] found that the more authentic students find the tasks that they are solving, the deeper the study approach they choose, which should result in enhanced learning.

Related to teachers' perceptions of assessment quality, Sach [66] analyzed the perceptions of 67 lower and middle school teachers of assessment. She found that teachers value formative assessment and believe it to enhance learning. However, teachers were less confident in actually employing formative assessment practices in their own courses.

These prior works suggest that it is imperative to make (formative) assessment tasks authentic [24, 76, 81], contextually relevant [76], and motivating [22]. Especially for the last two aspects, utilizing LLMs to personalize the context of the tasks to the interests of the student could be useful. Similarly, it is important to provide students with enough opportunities for practice in order for them to achieve mastery of the topic [81], which could be scaffolded by utilizing LLMs to generate programming exercises for formative assessment.

2.3 Large Language Models in Computing Education

Large language models (LLMs) have exhibited outstanding performance in many tasks, including code generation [9, 19, 20, 40, 41, 46, 63, 80, 89]. Recent advances have prompted active research on the use of LLMs in computing education [62]. There has been a number of studies focused on LLMs' ability to solve programming problems [19, 20, 63, 89]. Finnie-Ansley et al. discovered that even early versions of these models such as the now-deprecated Codex perform better than students in both CS1 and CS2 programming courses when it comes to code writing [19, 20]. Puryear and Sprint [63] and Wermelinger [89] have demonstrated that Github Copilot can generate coding solutions similar to those written by students. In addition to solving programming exercises and tests, LLMs have been found to successfully generate code explanations [43, 51, 70]. AI-generated code explanations were seen as helpful by learners [51] and clearer and more accurate than those created by students themselves [43].

Codex has also been assessed for its ability to generate explanations for programming error messages [45]. Although the results of the study by Leinonen et al. [45] were mixed, the authors have noted that such explanations could be used as a scaffold for understanding programming errors. Later works by Santos et al. [69] and Wang et al. [88] that used more recent models have demonstrated better performance. Santos et al. [69] found that providing the LLM the code that produced the error helped to enhance error messages. Wang et al. [88] discovered that students who utilized GPT-4-enhanced error messages repeated errors less frequently and required fewer attempts to fix them compared to those students who received traditional error messages.

LLMs have also been used to develop coding assistants for students, for instance, to power an on-demand tool providing support in undergraduate courses [47]. Liffiton et al. [47] argue that such tools are valuable as they offer instant help to students when they cannot get in touch with course instructors. According to the authors, an online system like theirs can also relieve students' anxiety about reaching out for help. Generative AI has also been investigated for generation of other sorts of learning materials, for example, worked examples. Jury et al. [37] have successfully implemented a tool for creating interactive worked examples. As the authors point out, developing such examples is time-consuming and LLMs can significantly facilitate their creation. Furthermore, GPT-3 has been used to boost gamification in an intelligent tutoring system [7]. The system was offered to a group of Chinese students in the UK with an intent to make their learning environment more inclusive and increase their sense of belonging. It was well received by the students who reported feeling supported [7].

2.4 Automatic Programming Exercise Generation

Prior to the emergence of modern generative AI models based on LLMs, template-based approaches were popular for automatic generation of programming exercises [61, 83, 84, 91]. In these template-based, parameterized exercises, certain parts of their problem descriptions are parameterized, meaning that specific parts of an exercise are modified according to defined parameters. This technically allows one to generate an infinite number of variations where, for example, numbers or specific statements in problem descriptions are different. As each student is presented with minor exercise differences, one typical use case for such exercises is to prevent plagiarism [64].

Using context free grammar to form program templates [1, 57] and generating random exercises from various models such as UML diagrams or mathematical notation [74, 79] have been suggested to improve purely parameter-based template approaches. These methods allow a way to individualize exercises for students. Such template-based approaches also make it possible to effectively personalize exercise content to themes and topics, as demonstrated by Zavala and Mendoza [91], who contextualized the generated exercises though linked open data. Nonetheless, they fail to generate completely new and varied exercises with deep personalization without extensive manual effort.

As LLMs displayed state-of-the-art performance in code generation and explanation tasks, Sarsa et al. [70] investigated the use of OpenAI Codex for generating programming exercise task descriptions with model solutions and automated tests. Their study yielded promising results, as the generated exercises were mostly sensible and sometimes of sufficient quality to be handed to students to solve without modification. However, roughly one fifth of the generated exercises did not make sense, and Codex sometimes struggled to accommodate concepts given as keywords to personalize the exercises. The authors additionally noted that Codex managed poorly in creating sample solutions and automated tests. On the other hand, focusing solely on solution and test generation, Chen et al. [8] achieved much better results by generating multiple solution and test pairs and then picking a sample where the generated solution passed the generated tests.

Replicating the work of Sarsa et al. [70] with a more modern AI model, namely GPT-4, del Carpio Gutierrez et al. [13] evaluated generated exercises for context relevance, description clarity, and problem sensibility in a larger study involving multiple prompting strategies. They assessed the generated content using rubrics and found the quality of the content to be high. Another replication of the work by Sarsa et al. [70] authored by Jordan et al. [36] explored the performance of GPT-3.5 in generating exercises in four natural languages. They found that problems generated in English, Spanish, and Vietnamese were mostly accurate and understandable and would only require minor modifications before giving them to students. However, the quality of the exercises generated in Tamil was poor, indicating that current models still do not completely generalize across natural languages.

In a similar work with a more manual approach, Speth et al. [75] generated programming exercise sheets with ChatGPT (GPT-3.5) chat sessions by having an instructor provide ideas and context to the model and then iteratively leveraging ChatGPT to refine the generated exercises within the ChatGPT session. While they noticed that ChatGPT was adept at quickly generating good exercises, they noted that instructors nearly always resorted to minor manual edits to improve exercise quality. Phung et al. [59] explored ChatGPT and GPT-4 against human tutors in generating debugging quizzes that could help students practice specific concepts (among other things). They focused on creating simplified versions of students' buggy pieces of code to help them practice solving bugs that they encounter, whereas our aim is to create new practice exercises contextualized to various themes and topics.

The unique contribution of our work is the focus on evaluating context personalization specifically. In addition, only one prior work on automatic exercise generation using LLMs actually had students complete the generated exercises – the study by Speth et al. [75] – and thus, more evidence on how students perceive LLM-generated exercises is needed.

3 METHODS

3.1 Prompt Engineering

The prompt used for generating the final exercise set went through multiple revisions both in terms of wording the prompt and the role of the request and determining an appropriate temperature value, which regulates randomness of the model's responses. The original version of the prompt only asked the model to produce a new exercise based on the example we provided and to contextualize it to a particular topic. This, however, resulted in the model often producing the same response when asked for an exercise contextualized to a popular theme, e.g., pets or outdoor activities. For example, when asking for an exercise related to pets, the model would almost solely generate exercises related to cats and dogs. To circumvent this limitation, we asked the model to list popular entities or things within certain categories, which constituted the themes used for personalization. The top level themes were arbitrarily chosen by the authors, whereas the topics within the themes were generated by the model. For instance, to obtain a list of topics for the *pets* theme, we asked the model to generate a list of ten popular pets. When asked to generate popular handicrafts, GPT-4 included home-brewing, which is why we had to explicitly forbid it from mentioning alcohol in order to keep the exercises suitable for a wide audience. Additionally, we tried to make the topics contextually relevant to our users by using such keywords as "Finland" and "the Nordic countries" in our requests. Our themes were the following: *outdoor activities, literature, historical landmarks, classical music, pop music, cartoons, food, pets, sports, video games, nature destinations, handicrafts, art, Christmas, party games, board games.* Additionally, we had to ask the model to omit words related to diseases, as it included *Pandemic* in the *board game* theme, which we considered unacceptable following the COVID-19 pandemic.

Once the model was given the topics to create new exercises, other issues appeared. When asked to generate a new problem about guinea pigs, it produced a response that contained incorrect arithmetical calculations and mentions of weight gain. Following this occurrence, we made a decision to restrict the model further and ask it to exclude trigger words associated with mental or physical disorders in order to keep the course materials inoffensive. The response related to guinea pigs was as follows.

> *'title': 'Guinea Pig Weight Gain', 'problemDescription': 'Write a program that calculates the weight gain of a guinea pig. The program should ask the user for the initial weight of the guinea pig and the amount of food it has eaten in grams. Then, it should calculate and print the weight gain of the guinea pig. If the user enters the initial weight as 500 grams and the amount of food eaten as 200 grams, the program should print the weight gain as 200 grams. Similarly, if the user enters the initial weight as 400 grams and the amount of food eaten as 300 grams, the program should print the weight gain as -100 grams.*

When determining an appropriate value for the temperature parameter, we gave the model the same prompt asking it to generate a new exercise about Mickey Mouse with two different temperature values. When the temperature value was set to 0.7, it produced the somewhat humorous problem description found below.

> *'title': 'Mickey Mouse Age Calculator', 'problemDescription': 'Mickey Mouse wants to know his age in dog years. Write a program that asks the user for Mickey Mouse's age in human years, and then calculates and prints his age in dog years. The conversion rate is 1 human year equals 7 dog years. The program should work as follows:"'10 70"''*

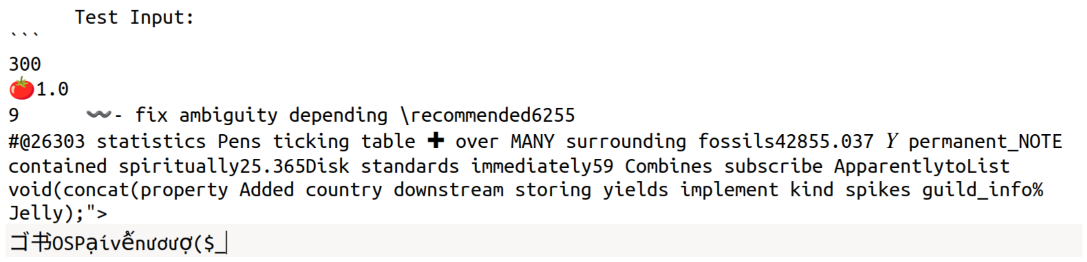
Lowering the temperature to 0.3 resulted in it completely ignoring the topic and producing an exercise unrelated to Mickey Mouse.

> *'title': 'Mickey Mouse Age Calculator', 'problemDescription': 'Write a program that asks the user for their birth year and the current year, and then calculates and prints their age. The program should work as follows:"'Enter your birth year: 1990 Enter the current year: 2022 Your age is: 32"''*

Raising the temperature to 1.5 led to the model hallucinating. Figure 1 contains a response with emojis.

Similarly, the temperature value of 2.0 resulted in another absurd response. Eventually, the temperature value of 0.5 was chosen to keep the model restrained to the structure of the examples we provided but allow for some variation in its output.

Additionally, the model was asked to include certain concepts covered in the first three chapters of the course. Two concepts were introduced in each chapter, and the model was instructed to use the concepts covered in the current and previous chapters. Without this restraint, the model would often produce exercises unrelated to the concepts discussed in the course. The first chapter covered *user input* and *program output*, the second one introduced *variables* and *arithmetics*, and the third one was dedicated to *conditional statements* and *logical operators.* To generate *normal* difficulty exercises, we asked the model to create new problems at the same



~~~~
        Test Input:
```
300
🍅1.0
9      〰- fix ambiguity depending \recommended6255
#@26303 statistics Pens ticking table ➕ over MANY surrounding fossils42855.037 𝑌 permanent_NOTE
contained spiritually25.365Disk standards immediately59 Combines subscribe ApparentlytoList
void(concat(property Added country downstream storing yields implement kind spikes guild_info%
Jelly);">
ゴ书osPạivễnưσượ($_
~~~~

Figure 1: A screenshot of nonsensical output containing emojis.

difficulty level as our examples. To make them a bit more difficult, we requested slightly more complex exercises. The model was instructed to avoid loops due to the limitations of the tool, which could potentially become stuck in an endless loop.

The final prompt is as follows (the variables in bold).

> *Please generate a short programming exercise in Dart based on the example that I will provide. It should be about* ***$theme****, specifically* ***$topic****. It should be at the same difficulty level as the example* ***/or*** *It should be slightly more complex than the example. It should mainly cover* ***$concept1*** *but can also include* ***$concept2****. Please follow the structure of the example and stay within its scope. You are allowed to include the following concepts in the new exercise:* ***$concepts****. Do not use loops. Your response should be a JSON string. Here is the example:* ***$example_exercise***

In order to provide blanket specifications to the model, we experimented with the role that is included in the GPT-4 request. The role mentioned that our students were programming novices, which was necessary so that the model would not produce exercises too advanced for an introductory course. We also repeated the structure of our desired output to prevent the model from deviating from it. To prevent copyright issues, the model was asked to avoid citing lyrics or literary works since our topics included music and book authors. Finally, as our students were not native English speakers, we asked the model to stick to simple English. Our final role included numerous instructions that the model was to follow in its output:

> *I want you to act as a programming teacher for an introductory Dart course. Your students are programming novices. I will provide some coding example exercises, and it will be your job to invent new ones. They should contain the following name-value pairs in JSON: title, problemDescription, exampleSolution, starterCode, tests. Your responses should be written in simple English. Do not cite music lyrics or books. Do not include any greetings, be concise. Do not mention trigger words associated with mental or physical disorders, for example, weight loss or diet.*

3.2 Exercise Generation

For the purposes of this study, we chose to pregenerate the exercises instead of having an on-demand system. This was done for multiple reasons. First, pregenerating the exercises allowed us to check them for material that could be offensive to a wide audience. This is a concern as LLM-generated content might include biases that were present in the training data of the LLM [18]. Second, possible outages in the availability of the LLM API could have caused technical disruptions in the system when students wanted to use the tool. Lastly, during prompt engineering we noticed that the LLM would sometimes generate faulty code (see Table 1), and in many cases this would have made exercises impossible to assess automatically (as they require working unit tests generated by the LLM).

The exercise generation was done with the aim to obtain a varied set of problems for each chapter covering different concepts. To achieve the desired variety, we originally created three exercises for each combination of difficulty (*normal* and *advanced*), concept (two concepts per chapter, six in total)[1], and theme (each theme contained 10 topics to choose from). We randomly picked three topics to generate novel exercises, and some were kept as spare in case the model failed to generate an exercise with working code for a particular combination. Originally, each chapter had 96 exercises[2]; however, some of them contained faulty test code – 25.0% from the first chapter, 20.8% from the second, 92.7% from the third (the detailed summary can be found in Table 1). The astounding number of broken exercises in the third chapter was caused by an extra space in the example exercise that was fed to the model and replicated in the generated instances. The issue was fixed manually. Some exercises had an issue with missing escape characters, which led to the final pool consisting of 93 exercises in the first chapter and 95 exercises in the remaining two chapters each. Although we experimented with explicitly asking the model to remember to use the escape character, it did not. After multiple requests, it seemed unlikely it would generate working code, which is why we stopped the exercise generation.

Although there were 16 themes used for exercise generation in total, some of them were closely related, for example, *sports* and *outdoor activities*. Due to this, we split the themes into three subsets. The first chapter contained *Christmas, classical music, food, historical landmarks, literature, party games, video games*, and *outdoor activities*. The second had *art, board games, cartoons, handicrafts, nature destinations, pets, pop music, sports* for the offered themes.

[1]user input, program output, variables, arithmetics, conditional statements, logical operators
[2]8 themes per chapter × 3 exercises per theme × 2 difficulty levels × 2 concepts = 96 exercises per chapter

Table 1: Summary of exercise generation per chapter.

Chapter	Percentage of exercises with faulty code	Final number of exercises
1	25.0%	93
2	20.8%	95
3	92.7%	95

The third chapter contained the mixture of the themes from the previous chapters: *literature, pop music, video games, party games, outdoor activities, handicrafts, arts, pets.* For each theme there were ten specific topics generated with the help of the model that were used in the prompt.

3.3 Study Context

The study was conducted in an open online introductory programming course that uses the Dart programming language and is offered by Aalto University in Finland. The course is worth 2 ECTS credit points, which corresponds to approximately 50 study hours. The course covers input and output, variables and arithmetics, conditionals and logical operators, looping, functions, and lists and maps. The course uses an online textbook platform. The online textbook has intertwined embedded theory and exercise parts with programming exercises and quizzes, where the exercises are automatically assessed by the platform. The programming exercises are completed in an embedded online integrated development environment (IDE), which is opened through the platform when students work on programming exercises. The IDE comes with normal IDE functionality, including syntax and error highlighting and the possibility to run the programs within the IDE; programming exercises are also submitted through the IDE.

As the course is an open elective online course, the course participants include both Aalto University students and lifelong learners. The platform does not distinguish between them, and as is usual for open online courses, attrition rates have room for improvement. Out of the students who complete at least one exercise in the first chapter of the course, 51% continue to the second chapter, and 44% continue to the third chapter.

The responsible teacher of the course did not provide access to demographic data for this study. However, when considering the demographic data, prior research on the course has highlighted that the participants come from a range of backgrounds [71]. Most of the participants who continue past the first chapters of the course are between 26-35 or 36-55 years old, have some experience from tertiary education, have taken no prior programming courses, participate in the course due to being interested in the topic, and self-estimate their programming knowledge as very low. The vast majority of the course population are lifelong learners as CS majors at our university have other mandatory introductory programming courses. Of those identifying their gender, approximately half of the participants self-identify as men, a bit more than one third as women, and the remaining either as other than men or women or do not wish to disclose their gender.

3.4 Tool

For the purposes of the study, we implemented a new component to the platform that allows retrieving LLM-generated programming exercises and showing them in the embedded IDE. When retrieving an exercise, students can select a theme, a concept, and a difficulty level. They can also allow that any of these are chosen randomly. Based on the selection, the platform then retrieves a problem description and starter code, which are shown to the student. Once the student has completed the exercise, they can submit the exercise for grading in a similar way as if they worked on the standard course exercises. The LLM-generated exercises were evaluated using automated tests that were also generated using an LLM.

The component was embedded into the first three chapters of the course: (1) input and output, (2) variables and arithmetics, and (3) conditionals and logical operators. The component was shown at the end of each chapter. The instructions before the component that the students saw were as follows (translated from Finnish).

> *The first chapters of the course offer AI-generated programming exercises. You can try these exercises below. Please choose a theme, a concept, and difficulty. After this, click the "Get Exercise"-button.*
>
> *At this point, the course platform will load a problem description and the exercise and show a programming environment where you can work on the exercise. Once you finish the exercise, we will ask you for feedback about the programming exercise.*
>
> *You can complete as many AI-generated programming exercises as you want. The exercises and problem descriptions are currently only available in English.*

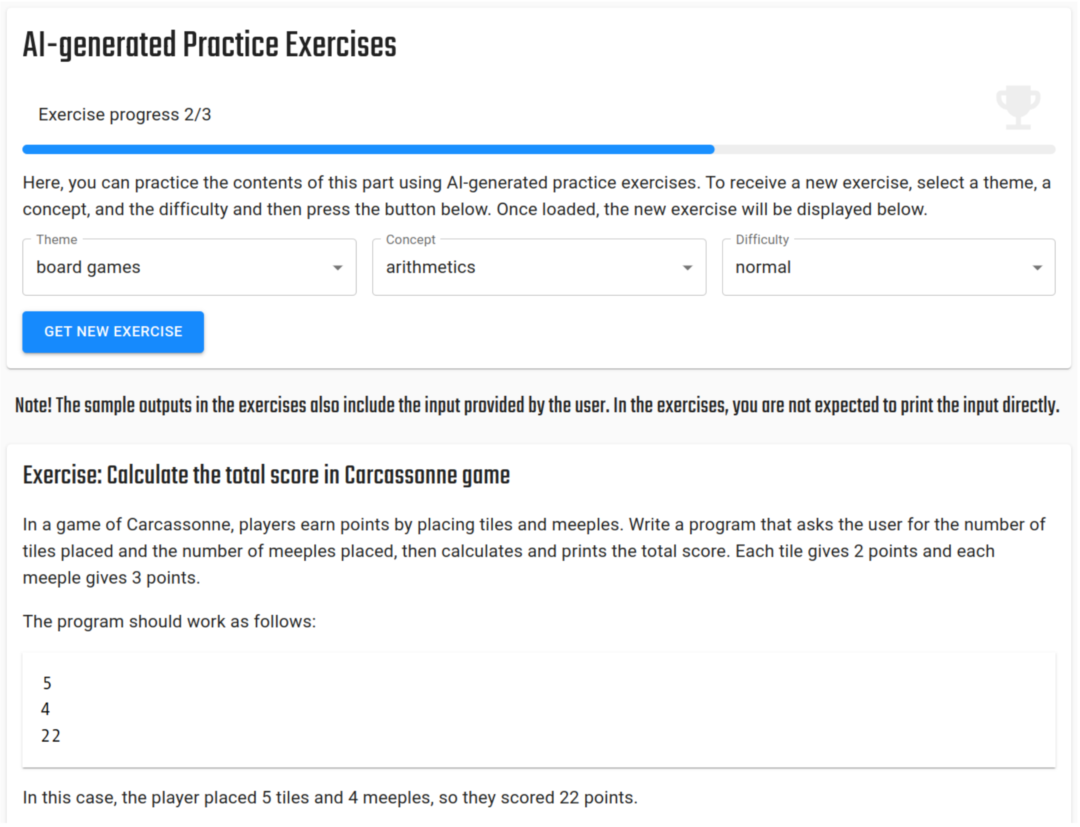
Completing the exercises did not yield any course points, and the students were not compensated in any other form. The tool had a progress bar that highlighted progress in the AI-generated exercises in each chapter. For each chapter, the progress bar filled after completing three exercises, at which point the students also saw a greyed out trophy icon being colored as yellow. This slight gamification was added to try nudge students into completing at least some of the AI-generated exercises.

Figure 2 below shows a screenshot of the component with the selection of the theme, the concept, and the difficulty, as well as a retrieved problem description.

3.5 Data Collection

The platform collected data on fetching exercises, where the data included a student identifier, a timestamp, the selections (theme, concept, difficulty), and the retrieved exercise. In addition, the platform collected data on the submissions, where the data similarly included the student identifier, a timestamp, an identifier for the exercise, and the submitted code. We also implemented survey functionality to the component. Once a student completed an exercise, they were shown a survey regarding the exercise, with the following four questions.

(1) The exercise description was clear.
(2) The exercise description matched the selected theme.
(3) The exercise description matched the selected concept.


AI-generated Practice Exercises

Exercise progress 2/3

Here, you can practice the contents of this part using AI-generated practice exercises. To receive a new exercise, select a theme, a concept, and the difficulty and then press the button below. Once loaded, the new exercise will be displayed below.

Theme: board games
Concept: arithmetics
Difficulty: normal

Note! The sample outputs in the exercises also include the input provided by the user. In the exercises, you are not expected to print the input directly.

Exercise: Calculate the total score in Carcassonne game

In a game of Carcassonne, players earn points by placing tiles and meeples. Write a program that asks the user for the number of tiles placed and the number of meeples placed, then calculates and prints the total score. Each tile gives 2 points and each meeple gives 3 points.

The program should work as follows:

```
5
4
22
```

In this case, the player placed 5 tiles and 4 meeples, so they scored 22 points.

Figure 2: A screenshot of the exercise selection functionality with a problem description shown. In the screenshot, the user has already completed two exercises. The user has selected "board games" as the theme, "arithmetics" as the concept, and "normal" as the difficulty, and then pressed the "Get Exercise"-button. The button is labeled as "Get New Exercise" as an exercise has already been retrieved. In this example, an exercise about Carcassonne has been retrieved.

(4) The exercise difficulty matched the selected difficulty.

Once a student completed three exercises within a chapter, they were shown a different survey with the following questions.

(1) The exercises were useful for my learning.
(2) The exercises were engaging to me.
(3) The theme selection was satisfactory to me.
(4) I enjoyed being able to select themes that match my interests.
(5) Please provide open feedback on the AI-generated practice exercises.

With the exception of the last question ("Please provide open feedback on the AI-generated practice exercises."), the questions were given as 7-item Likert-scale questions with the following prompt: *Please indicate how much you agree with each of the following statements, or how true it is about you. Use the scale from 1 to 7, where 1 is 'Strongly disagree' and 7 is 'Strongly agree'.*

Responding to the surveys was optional – students could skip the surveys if they wished. Data for this study was collected over four weeks in March 2024.

3.6 Approach

3.6.1 RQ1: Expert Evaluation by the Study Authors. For research question 1, *"How do the study authors evaluate the quality of contextually personalized exercises generated by GPT-4?"*, we follow the methods of prior work that has evaluated LLM-generated exercises by developing a rubric for quality assessment of exercises [13, 36, 70]. The rubric questions can be found in Table 2.

Out of the 283 programming exercises, 33 were randomly selected for inter-rater reliability analysis. The five authors discussed the rubric and rated the 33 exercises according to the rubric. The results of the inter-rater reliability analysis are presented in Table 2. We report both the percentage agreement and Gwet's AC1 statistic. We chose Gwet's AC1 as the inter-rater reliability metric due to the limitations of other multi-rater IRR statistics, such as Krippendorf's alpha and Fleiss' kappa, caused by high agreement between raters [16, 17, 27, 28].

After the IRR analysis, each author individually evaluated a set of 50 randomly selected exercises. Thus, all the exercises that were included in the tool were assessed in the expert evaluation. We report the exact numbers of answers for each question as well as percentages.

For the final set of evaluations, for the 33 exercises that were evaluated in the IRR analysis, we use the majority vote value. There were only four ties for majority value, all of them occurring for the "shallow vs. deep personalization" question. For three ties, they were between "unsure" and "deep" and for one, between "unsure" and "shallow". In these four cases, we chose the other option than "unsure" for the final evaluation set.

3.6.2 RQ2: Student Evaluation. For research question 2, *"How do students evaluate the quality of contextually personalized exercises generated by GPT-4?"*, we analyze the feedback on the generated exercises provided by the course participants.

We report the distribution of the Likert-answers to the survey questions, which are listed in Section 3.5. Due to the low number of responses to the open feedback question (n = 4), we do not analyze it.

3.6.3 RQ3: Student Interactions. For research question 3, *"How do students interact with the tool that provides contextually personalized exercises?"*, we report how many exercises were fetched and solved in addition to the number of the course participants who solved all three exercises necessary for obtaining a trophy. We analyzed what choices they made when fetching exercises in terms of the theme and difficulty selection. Since their activity levels differed, we calculated theme popularity as well as their' preference for random theme selection as an average ratio for the normalized number of exercises retrieved per student (i.e., for each user, we calculated how often they selected a specific vs. a random theme as a percentage). We additionally report how much time passed between each exercise retrieval and its solution being submitted per exercise. This is done to describe the impact of exercise difficulty levels on student performance as measured by time-on-task, which has been shown to strongly correlate with performance [42]. Using the expert evaluation of the exercises, we analyze how unclear exercise descriptions, inclusion of advanced concepts, and inadequate difficulty correlated with student performance.

4 RESULTS

4.1 RQ1: Expert Evaluation by the Study Authors

The results of the inter-rater reliability analysis for the expert evaluation are presented in Table 2. From the table, it can be seen that the level of agreement varied between the questions. Agreement was quite high for the first five questions but was lower for the last two.

The results of the expert evaluation of the 283 generated exercises is shown in Table 3. Answering the first question, *"The exercise description was clear"*, we considered the overwhelming majority of the exercises to be clear. As for the second, *"The exercise description matched the selected theme"*, and third, *"The exercise description matched the selected topic"*, questions, we found that almost all of them matched both their theme and topic. When evaluating the exercises for the fourth question, *"The exercise description matched the selected concept"*, we concluded that 87.6% of them corresponded to their requested concept. When it comes to assessing whether the exercises included concepts that were too advanced (the fifth question), we found that most of them did not contain concepts that were out of scope for the corresponding chapter of the course. However, when evaluating whether the exercise difficulty matched the selected difficulty (the sixth question), we concluded that the difficulty of the exercises was frequently not satisfactory. We found that the difficulty was rated as "too easy" in 39.6% of the cases, "too difficult" in 6.0%, and "okay" in 54.4% of the exercises. As for the depth of personalization (the seventh question), it was shallow in the majority of the generated exercises (64%), while the rest were either somewhere in between (9.5%) or deeply personalized (26.5%).

The cases where there were too difficult concepts included occurred mostly in the first chapter. In these cases, when the model was asked for an "advanced" exercise related to user input or output, it would produce an exercise requiring conditionals (introduced in Chapter 3), e.g., print different outputs depending on user input. See Figure 3 for an example where the concept was "user input" (Chapter 1), but the exercise required conditionals to solve.

Figure 3: An exercise example with overly advanced concepts.

Write a program that asks the user for their favorite historical landmark. If the user's favorite is the Tower of London, the program should print a message to the user 'Tower of London is a great choice!', where Tower of London is the landmark entered by the user. For example, with the input 'Tower of London', the program output is as follows:

```
```
What is your favorite historical landmark?
Tower of London
Tower of London is a great choice!
```
```

However, if the user enters a different landmark, the program should simply print the name of that landmark.

Related to matching the theme and topic, we found that sometimes the problem description would match them but contain some factual errors. See Figure 4 for an example. In the example, it is claimed that in Pictionary, teams score between 0 and 3 points depending on whether their guess is correct, almost correct, wrong, or not guessing at all. In actuality, teams just score a single point for a correct guess[3].

As mentioned above, most of the exercises were only shallowly personalized. Often, there was a sentence or two about the theme or topic in the problem description, but the actual exercise was not directly relevant to the context. See Figure 5 for an example of a shallowly personalized exercise. The theme for the exercise was "pop music", and its topic was "Ariana Grande". However, only the first sentence of the exercise mentions Ariana Grande and album sales, while the rest of it is not related to the theme or topic. There were some deeply personalized exercises too (for example, Figure 6). In the exercise, most of the problem description deals with the theme ("art") and the topic ("Pablo Picasso"), and the task is directly relevant to the topic.

[3]As per the rules: https://service.mattel.com/instruction_sheets/T5132-0920.pdf

Table 2: Percentage agreement and Gwet's AC1 for expert evaluation.

Question	Answer Options	% Agreement	Gwet's AC1
1. The exercise description was clear	Yes/Partially/No	79%	0.90
2. The exercise description matched the selected theme	Yes/Partially/No	94%	0.97
3. The exercise description matched the selected topic	Yes/Partially/No	85%	0.94
4. The exercise description matched the selected concept	Yes/No	100%	1.00
5. Included concepts that were too advanced	Yes/No	94%	0.96
6. The exercise difficulty matched the selected difficulty	Too easy/Okay/Too difficult	27%	0.49
7. Shallow vs. deep personalization	Deep/Unsure/Shallow	18%	0.34

Table 3: Summary of expert evaluation.

Question	Response	Count	Percentage
1. The exercise description was clear	Yes	273	96.5%
	Partially	10	3.5%
	No	0	0.0%
2. The exercise description matched the selected theme	Yes	272	96.1%
	Partially	7	2.5%
	No	4	1.4%
3. The exercise description matched the selected topic	Yes	270	95.4%
	Partially	9	3.2%
	No	4	1.4%
4. The exercise description matched the selected concept	Yes	248	87.6%
	No	35	12.4%
5. Included concepts that were too advanced	Yes	14	4.9%
	No	269	95.1%
6. The exercise difficulty matched the selected difficulty	Too easy	112	39.6%
	Okay	154	54.4%
	Too difficult	17	6.0%
7. Shallow vs. deep personalization	Deep	75	26.5%
	Unsure	27	9.5%
	Shallow	181	64.0%

There were a few cases where the problem description matched the topic (which was not visible to the course participants) but not the theme (which was chosen by the student). See Figure 7 for an example. Here, the theme was "Christmas", and the topic was "baking cookies". The exercise is clearly relevant to the topic but not directly relevant to the theme (and would probably better fit the theme of "Cooking", for example).

In many cases, particularly for the arithmetic exercises, their problem descriptions contained deeply personalized details, but the corresponding code solutions were strikingly similar to the example problems we provided to the model. For instance, the solution to the problem in Figure 8 could be solved with exactly the same code containing a subtraction operation as the example exercise (Figure 9) given to the model. However, the exercise could be considered deeply personalized as calculating the difference in goals is relevant to the theme ("sports") and the topic ("handball").

4.2 RQ2: Student Evaluation

Figure 10 shows the results of the student evaluation of the generated exercises. Figure 10a shows the distribution of the Likert-scale responses to the survey given after each exercise and Figure 10b shows the distribution for the survey shown to the students after they had completed three exercises. There were a total of 79 responses to the exercise-specific survey and 28 responses to the general survey shown after completing three exercises.

From the figures, it can be seen that the student feedback was overwhelmingly positive, with over half of the students strongly agreeing with every statement. The five statements that did not receive any disagreements are the following: *matched selected theme, matched selected concept, useful for my learning, engaging to me,* and *theme selection was satisfactory.* The three statements that received negative responses (*description was clear, matched selected difficulty,* and *enjoyed being able to select themes*) only had a few responses disagreeing with the statements (12%, 10%, and 3% respectively).

4.3 RQ3: Student Interactions

Tables 4 and 5 contain summaries of descriptive statistics for the interaction data collected by the tool. Table 4 shows the overall number of the students who fetched and solved exercises and the range, mean, median, and standard deviation for the fetched and solved exercises. Table 5 shows the number of the fetched and solved exercises and the number of the individual course participants split

Figure 4: An exercise containing a factual error.

In a game of Pictionary, each team gets a score between 0 and 3 for each round, based on the following scale:

```
<table>
<tr>
<th>Score</th>
<th>Result</th>
</tr>
<tr>
<th>3</th>
<th>Correct Guess</th>
</tr>
<tr>
<th>2</th>
<th>Almost Correct</th>
</tr>
<tr>
<th>1</th>
<th>Wrong Guess</th>
</tr>
<tr>
<th>0</th>
<th>No Guess</th>
</tr>
</table>
```

Write a program that asks the user for a score and prints the result related to that score. If the user enters any other score, the program should print the message <code>Invalid Score!</code>.

Below is an example of the expected operation of the program.

```
<pre>
What score?
<b>&lt; 2</b>
Almost Correct
</pre>

Another example.

<pre>
What score?
<b>&lt; 4</b>
Invalid Score!
</pre>
```

Figure 5: An exercise with shallow personalization.

Write a program that asks the user for the number of albums Ariana Grande sold in two different years. Then, print the difference between them. If the user enters the numbers 3 million and 2 million, the program should print the number 1 million. Similarly, if the user enters the numbers 2 million and 3 million, the program should print the number -1 million.

The program should work as follows:

~~~~
```
3
2
1
```
```
2
3
-1
```
~~~~

Figure 6: An exercise with deep personalization.

Pablo Picasso had different periods in his art career. One of them is the Blue Period from 1901 to 1904. During this period, he painted essentially monochromatic paintings in shades of blue and blue-green. Write a program that asks the user for a year and prints whether or not it falls within Picasso's Blue Period. If the user enters any other year, the program should print the message <code>Not in the Blue Period!</code>.

Below is an example of the expected operation of the program.

```
<pre>
Which year?
<b>&lt; 1902</b>
Yes, in the Blue Period.
</pre>
```

Another example.

```
<pre>
Which year?
<b>&lt; 1900</b>
Not in the Blue Period!
</pre>
```

by chapter. Note that the same person might have used the tool across multiple chapters, which is why the number summed in Table 5 is higher than the number of the students in Table 4. A total of thirty seven users solved three or more exercises, another two solved two exercises, and the remaining eight submitted one correct solution.

Table 5 shows that the majority of the recorded interactions with the tool belonged to the first chapter. Since the course was self-paced and with a high attrition rate, there were fewer users in the latter chapters of the course. The tool included the following distinct themes in the first chapter menu: *Christmas, classical music, food, historical landmarks, literature, party games, video games* and *outdoor activities*. Since some of the users were more active than

Figure 7: An exercise relevant to the topic but not to the theme.

Write a program that asks the user for the number of cookies they want to bake. After this, the program prints a message to the user 'You will bake N cookies!', where N is the number entered by the user. For example, with the input '5', the program output is as follows:

~~~~
```
How many cookies do you want to bake?
5
You will bake 5 cookies!
```
~~~~

Similarly, if the user enters the number '12', the program output is as follows:

~~~~
```
How many cookies do you want to bake?
12
You will bake 12 cookies!
```
~~~~

Figure 8: An exercise that can be solved using the same code as the exercise that was provided to the model as an example.

Write a program that asks the user for the number of goals scored by two different teams in a handball match, and then prints the difference between them. If team A scored 10 goals and team B scored 3 goals, the program should print the number 7. Similarly, if team A scored 4 goals and team B scored 8 goals, the program should print the number -4.

The program should work as follows:

~~~~
```
12
15
-3
```
~~~~

Table 4: Summary of student interaction data (for those students who fetched at least one exercise).

	N	Range	Mean	Median	SD
Fetched	68	[1, 27]	5.32	3	5.55
Solved	47	[0, 10]	2.87	3	2.88

the others, the theme popularity shown in Figure 11 is the mean ratio of ratios per each individual user. According the Figure 11, *food* and *video games* were the most engaging to the course participants. The second and third chapters saw a decline in the user interaction. The second chapter offered the following set of themes: *art, board games, cartoons, handicrafts, nature destinations, pets, pop music,* and *sports*. In the second chapter, the popular choices were *board games, cartoons, nature destinations* and *art*. The third chapter exhibited the same trend as the second one, which is a further decline in the student engagement with the tool. *Video games*, a well-liked theme in the first chapter, was also the most popular among the users in the third chapter. Table 6 indicates that the users preferred choosing individual themes over random ones across all the chapters, which is also reflected in their feedback responses. While the context personalization of the exercises was well received by the users, Table 6 indicates that this was not the case for the concept personalization. The study participants displayed a preference for particular concepts in the first chapter; however, they mostly picked random concepts in the second and third chapters. During the expert evaluation, we noticed that the choice of concepts did not appear to significantly affect the variability in the generated exercises, which could have reduced the students' interest in selecting specific concepts . For example, most of the exercises on "user input" also involved "output" and vice versa, essentially making the choice between them meaningless.

Figure 9: One of the three example exercises used as input.

Write a program that asks the user for two numbers, and then prints the difference between them. If the user enters the numbers 10 and 3, the program should print the number 7. Similarly, if the user enters the numbers 4 and 8, the program should print the number -4.

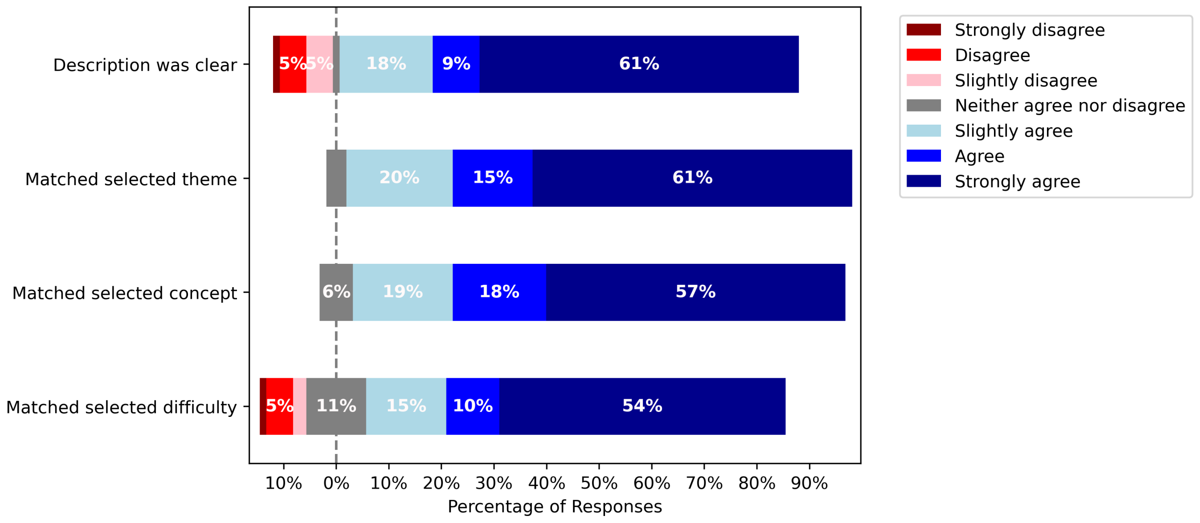
The program should work as follows:

~~~~
```
12
15
-3
```
~~~~

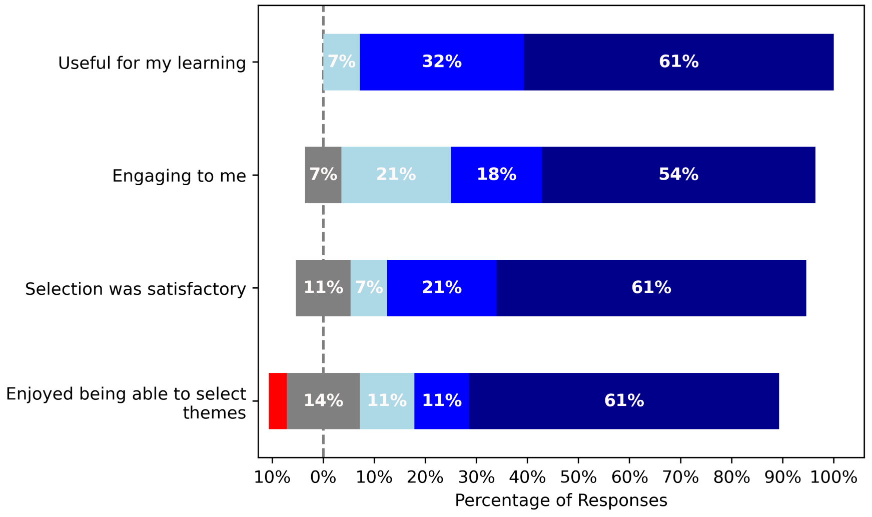
For the solved exercises, we investigated how much time passed between fetching an exercise and submitting its correct solution. There were 4 cases of the users fetching the same exercise twice but solving it once[4]. We concluded that the retrieval closest to the submission is the most plausible and discarded the other event. Additionally, there were a few outliers where it took some students over an hour (up to 2 days) to submit their solutions. They were omitted, as it is unlikely those students actually worked on the exercises that long and instead might have taken a break between fetching and solving the exercises. Figure 12 shows that the users generally did not require more time for the advanced exercises and on average could solve one problem in less than 10 minutes.

Table 7 illustrates the differences between the exercises that were solved and unsolved in terms of difficulty, clarity, and presence of advanced concepts. The exercises' difficulty corresponds to the generated difficulty, whereas the other statistics (clarity and

[4]This could happen because skipped exercises were returned to the pool.



(a) Student evaluation results (per exercise).



(b) Student evaluation results (after three exercises).

Figure 10: Student evaluation results. (a) shows the responses to the questionnaire that was shown after each exercise and (b) shows the responses to the questionnaire that was shown after three completed exercises.

Table 5: Summary of student interactions with the tool split by chapter.

Chapter	# fetched exercises	# solved exercises	# users who retrieved exercises	# users who solved exercises
1	177	106	53	40
2	108	60	29	22
3	71	28	20	11

presence of advanced concepts) are taken from the expert evaluation. The unsolved exercises contain more occurrences of only partially clear problem descriptions and advanced concepts that were not covered in the course materials. Table 8 shows that in the first chapter 12% of the unsolved exercises were too difficult and 10% in the second. Additionally, 57% of the solved exercises in the third chapter consisted of the problems that were too easy for their declared difficulty level.

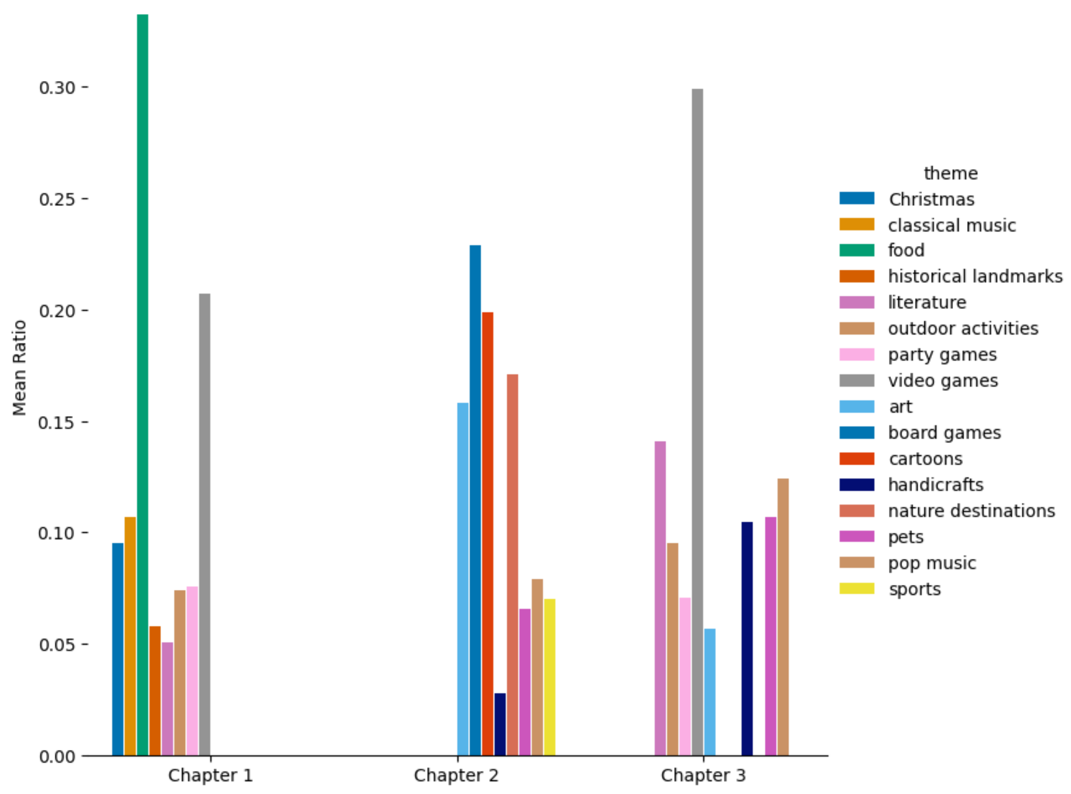


Figure 11: Theme popularity per chapter.

Table 6: Comparison of random vs. specific context and concept preferences.

		Theme choice		Concept choice	
Chapter	Type	Random	Specific	Random	Specific
1	Fetched	22%	78%	35%	65%
	Solved	22%	78%	35%	65%
2	Fetched	42%	58%	67%	33%
	Solved	40%	60%	47%	53%
3	Fetched	35%	65%	55%	45%
	Solved	41%	59%	56%	44%

5 DISCUSSION

Both the expert and student evaluations indicate that the quality of the generated exercises was high. In addition, they were well received by the students. This suggests that AI-generated problems could be a valuable addition to introductory programming courses, at least, as additional practice material as was the case in this study. One reason for the positive student feedback could be that the context personalization made the tasks feel more authentic to them, which has been found to positively impact students' perceptions of assessment [22, 24, 76]. These results support earlier findings by Sarsa et al. [70], Jordan et al. [36], and del Carpio Gutierrez et al. [13] who also found the quality of generated exercises to be generally high. The quality of the exercises was especially impressive considering that the language of the course was Dart, which is not as popular as many other languages; thus, the model is likely to have seen less Dart code during training compared to more common languages, such as Python, Java, or C.

One potential downside of our approach is that GPT-4 failed to produce content according to the specified difficulty level, e.g., some was too easy or included concepts that were too advanced. Thus, we suggest that this approach is best suited for providing optional practice opportunities that would complement instructor-crafted programming exercises. On the other hand, many shallowly personalized exercises could be solved with practically identical code; consequently this automated approach to exercise generation could be used for circumventing plagiarism. Students could be provided with test problems containing minor variations, as prior work has found little effect of contextualization on programming problem solving performance [6, 48]. Examining the generated exercises from the point of view of the depth and granularity of personalization, our results indicate that the majority (64.0%) included shallow personalization, especially when the model was severely limited in the number of concepts it could use, as was the case for Chapter 1. Since practically all the exercises were tailored to their narrow topic, we conclude that the granularity of the personalization was fine in the vast majority of the cases. This suggests that our approach is suitable for producing mostly shallow context personalization;

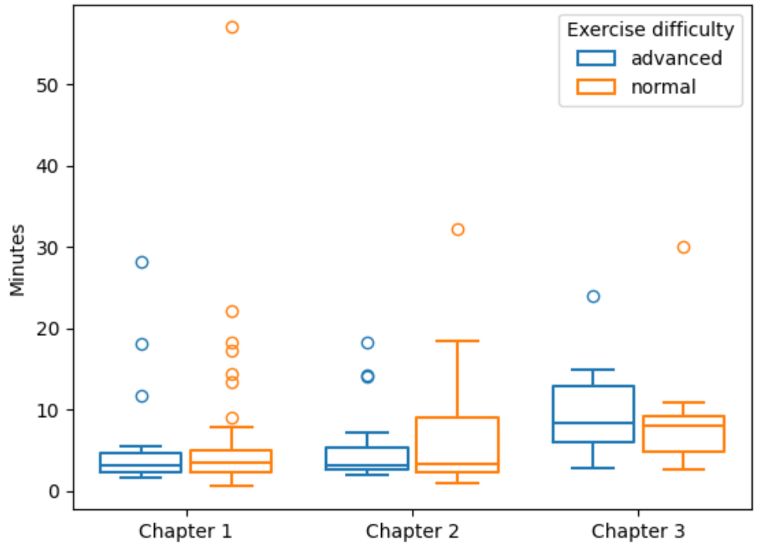


Figure 12: Distribution of time between fetching and solving an exercise per each individual user and exercise combination.

Table 7: Summary of exercise results.

Chapter	Exercise difficulty		Clarity		Advanced concepts	
Unsolved Exercises						
1	Normal	66%	Clear	96%	Absent	83%
	Advanced	34%	Partially clear	4%	Present	17%
2	Normal	65%	Clear	90%	Absent	92%
	Advanced	35%	Partially clear	10%	Present	8%
3	Normal	82%	Clear	97%	Absent	100%
	Advanced	18%	Partially clear	3%	Present	0%
Solved Exercises						
1	Normal	66%	Clear	98%	Absent	100%
	Advanced	34%	Partially clear	2%	Present	0%
2	Normal	58%	Clear	93%	Absent	100%
	Advanced	42%	Partially clear	7%	Present	0%
3	Normal	50%	Clear	100%	Absent	100%
	Advanced	50%	Partially clear	0%	Present	0%

Table 8: Difficulty match of solved and unsolved exercises.

Chapter	Solved exercises			Unsolved exercises		
	Too easy	Okay	Too difficult	Too easy	Okay	Too difficult
1	22%	78%	0%	24%	64%	12%
2	33%	65%	1%	27%	63%	10%
3	57%	43%	0%	30%	68%	2%

however, it is possible to obtain problems of the fine grain size of personalization by providing a model with a specific topic of interest.

The results suggest that using LLMs for contextual personalization of exercises is meaningful for stimulating learners' engagement and potentially preventing boredom, which is often reported by students in computing education [11]. In the optional feedback survey (Figure 10b), most of the students agreed that being able to select a theme was enjoyable. What is more, both the expert and student evaluation found that the majority of the exercises matched their declared theme (Figure 10a and Table 3). The interaction data supported this by showing that the course participants were more likely to choose a specific theme over a random one (Table 6) and clearly displayed predilections for particular themes (Figure 11). Since there is an apparent preference for contextually relevant exercises, providing them could lead to increased engagement and motivation among learners and higher course participant retention [6, 15, 25, 26]. Improved engagement with exercises, in turn, could lead to higher time-on-task, which has been found to be a good predictor of performance in introductory programming [42]. Because the exercise tool employed a light gamification element in the form of a trophy given for solving three exercises per chapter, such repetitive engagement could also help students' test scores, as has been demonstrated by Edwards et al. [15]. Moreover, the optional student feedback indicated that the course participants positively evaluated the utility of the exercises for their learning (Figure 10b). This result is in line with earlier research on hands-on approaches to computing education, which have been perceived as beneficial by students [50, 77].

Additionally, contextually personalized exercises may contribute to enhanced situational interest that can potentially lead to the more advanced phases of interest development, i.e., emerging and well-developed individual interest stages [5, 33–35, 54]. For example, research in mathematics education by Høgheim and Reber [35] has indicated that shallow context personalization positively affects students with low individual interest in the subject by supporting their situational interest. Since the majority of the generated problems were shallowly personalized (Table 3), this effect could potentially translate to computing education students. For those students who have already achieved the initial stage of individual interest development, i.e., emerging individual phase, such exercises can help maintain their interest by serving as an additional source of learning materials. As some of the exercises were personalized to the cultural background of the study participants, they could increase learners' sense of intrinsic and attainment values, as was demonstrated by Schoenherr [72].

5.1 Limitations

There are some limitations to this study, which we outline here. First, the course was an elective, self-paced, online course, where the participants were mostly lifelong learning students (i.e., not formally enrolled at a university). Moreover, earlier empirical research on contextually personalized learning materials has been mostly limited to mathematics in secondary education [5, 34, 35, 72, 85, 86]. Thus, it is uncertain whether the results found here would generalize to more traditional introductory programming courses with deadlines, where the majority of participants are computer science or other STEM majors. Such students might be more motivated to complete such courses in general compared to lifelong learners who constitute the majority of the course population. The experiment was also done at a single institution, which also limits its generalizability.

Another limitation is the lack of background information concerning the preexisting levels of interest and competence in programming among the course participants. Earlier research on context personalization has shown that those students who possess less interest and lower self-perceived competence in a subject, e.g., mathematics, benefit more from contextually personalized problems [34, 35]. In their case, context personalization triggers situational interest. What is more, the depth of personalization affects various categories of learners in a different fashion, e.g., those who engage more with mathematics through their interests are more positively influenced by deep personalization in mathematical problems [86]. Since we did not obtain any detailed data on the course participants' prior engagement and attitudes towards computer programming, we could not assess how different levels of personalization affected their study progress or feedback on the exercises. Additionally, we are uncertain to which extent contextualization might have influenced their situational or individual interest in the subject area.

As the course was offered entirely online and there was no contact between the study authors and the study participants, the risk of subject and experimenter expectancy biases [23, 93] could be considered small due to these effects being primarily evidenced in interpersonal settings [30, 39, 65]. Nonetheless, we acknowledge that, apart from the open feedback, all of our survey questions were Likert-scaled and positively inclined towards effects desired by the authors, e.g., "The exercises description matched the selected theme" and "I enjoyed being able to select themes that match my interests". This makes our surveys susceptible to an acquiescence bias, where responders tend to passively agree (or disagree) to asked questions irrespective of content [60]. However, both age and education have been observed to mitigate the effect of acquiescence on questionnaires [10], and as the course participants in general are lifelong learners and university students, both the average age and education among the participants was likely to be reasonably high. Besides the phrasing of the questions themselves, the questionnaires or the shown exercises did not contain anything that could be interpreted as suggestive.

Related to the generated exercises, we opted to pregenerate the exercises to avoid running into any problems with an on-demand system, such as the LLM being unavailable for some reason or students trying to do prompt injection attacks to break the tool [58]. The downside of this approach was that the theme selection was also predefined and thus might not have matched the students' top interests. However, multiple varying themes were available, and we were mostly interested in seeing if students were keen to select any particular theme over a random one. Another limitation of having the exercises pregenerated is that the exercise pool was limited, and consequently the situation did not exactly mimic one where students would have truly unlimited practice opportunities.

Completing the exercises was optional, and no course credit was provided for completing them. This may have caused a selection bias, as active students might have used the tool more frequently. Since we openly told the course participants that the exercises were AI-generated, it is also possible that those students who are interested in AI were more likely to complete them, potentially skewing the results of the surveys. Such students might have also rated the AI-generated content more favorably compared to those learners who have more neutral or negative views on AI.

Related to the expert evaluation, two questions had relatively low agreement scores: determining if the exercise difficulty matched the selected difficulty and whether the personalization was deep or shallow. Thus, the results utilizing the expert evaluation data for these two questions should be taken with a grain of salt – the other evaluators could have made different decisions on these questions.

One limitation of the experimental design was that the course was organized in Finnish, but the tool, surveys, and generated exercises were in English. We opted to generate exercises in English as prior work has found that LLMs perform the best in English [92]. Recent work exploring the generation of exercises with LLMs in other languages besides English has demonstrated promising results [36], which is why we are interested in implementing the tool in Finnish in the future. The language of the exercise descriptions could have affected students' ratings as they are not native English speakers; however, Finns generally possesses good English skills.

6 CONCLUSIONS

In this work, we explored how successfully LLMs, namely GPT-4, can generate programming exercises. The generated content was offered for additional student practice in an online programming course. We evaluated the quality of the generated exercises by having both the authors and the students in the course assess them. In addition, we examined how the course participants interacted with the exercises in the course e-book.

Our findings indicate that the vast majority of the exercises generated by the LLM were clear and matched various themes, topics, and concepts. What is more, they rarely included concepts that were too advanced for the students in the course. However, the exercises were often easier than requested in the prompt given to the model, and the thematic personalization was often shallow.

In our future work, we are interested in studying how adding gamification features to the tool affects student engagement. Additionally, we are interested in examining how students' programming experience correlates with their interaction with LLM-generated exercises. We are working on a version of the tool that would allow students to generate exercises related to any contextual theme on demand, instead of having a list of predefined themes available in the tool.

ACKNOWLEDGMENTS

This research was supported by the Research Council of Finland (Academy Research Fellow grant number 356114).

REFERENCES

[1] Abejide Ade-Ibijola. 2019. Syntactic Generation of Practice Novice Programs in Python. In *ICT Education: 47th Annual Conference of the Southern African Computer Lecturers' Association, SACLA 2018, Gordon's Bay, South Africa, June 18–20, 2018, Revised Selected Papers 47*. Springer, 158–172.

[2] Joe Michael Allen, Frank Vahid, Kelly Downey, and Alex Daniel Edgcomb. 2018. Weekly Programs and in a CS and Class: Experiences and with Auto-graded Many-small and Programs (MSP). In *2018 ASEE Annual Conference & Exposition*.

[3] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. ACM. https://doi.org/10.1145/3545945.3569759

[4] Matthew L. Bernacki, Meghan J. Greene, and Nikki G. Lobczowski. 2021. A Systematic Review of Research on Personalized Learning: Personalized by Whom, to What, How, and for What Purpose(s)? *Educational Psychology Review* 33, 4 (2021), 1675–1715. https://doi.org/10.1007/s10648-021-09615-8

[5] Matthew L. Bernacki and Candace Walkington. 2018. The role of situational interest in personalized learning. *Journal of Educational Psychology* 110, 6 (2018), 864–881. https://doi.org/10.1037/edu0000250

[6] Dennis Bouvier, Ellie Lovellette, John Matta, Bedour Alshaigy, Brett A. Becker, Michelle Craig, Jana Jackova, Robert McCartney, Kate Sanders, and Mark Zarb. 2016. Novice Programmers and the Problem Description Effect. In *Proceedings of the 2016 ITiCSE Working Group Reports (ITiCSE '16)*. ACM. https://doi.org/10.1145/3024906.3024912

[7] Chen Cao. 2023. Scaffolding CS1 Courses with a Large Language Model-Powered Intelligent Tutoring System. In *28th International Conference on Intelligent User Interfaces (IUI '23)*. ACM. https://doi.org/10.1145/3581754.3584111

[8] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397* (2022).

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *ArXiv* abs/2107.03374 (2021). https://api.semanticscholar.org/CorpusID:235755472

[10] Shane Costello and John Roodenburg. 2015. Acquiescence response bias—Yeasaying and higher education. *The Educational and Developmental Psychologist* 32, 2 (2015), 105–119.

[11] Mayela Coto, Sonia Mora, Beatriz Grass, and Juan Murillo-Morera. 2021. Emotions and programming learning: systematic mapping. *Computer Science Education* 32, 1 (2021), 30–65. https://doi.org/10.1080/08993408.2021.1920816

[12] Michelle Craig, Jacqueline Smith, and Andrew Petersen. 2017. Familiar contexts and the difficulty of programming problems. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling 2017)*. ACM. https://doi.org/10.1145/3141880.3141898

[13] Andre Del Carpio Gutierrez, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating Automatically Generated Contextualised Programming Exercises. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM. https://doi.org/10.1145/3626252.3630863

[14] Jacquelynne S. Eccles and Allan Wigfield. 2020. From expectancy-value theory to situated expectancy-value theory: A developmental, social cognitive, and sociocultural perspective on motivation. *Contemporary Educational Psychology* 61 (2020), 101859. https://doi.org/10.1016/j.cedpsych.2020.101859

[15] John Edwards, Joseph Ditton, Dragan Trninic, Hillary Swanson, Shelsey Sullivan, and Chad Mano. 2020. Syntax Exercises in CS1. In *Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER '20)*. ACM. https://doi.org/10.1145/3372782.3406259

[16] Rosa Falotico and Piero Quatto. 2015. Fleiss' kappa statistic without paradoxes. *Quality & Quantity* 49 (2015), 463–470.

[17] Alvan R Feinstein and Domenic V Cicchetti. 1990. High agreement but low kappa: I. The problems of two paradoxes. *Journal of clinical epidemiology* 43, 6 (1990), 543–549.

[18] Emilio Ferrara. 2023. Should ChatGPT be biased? Challenges and risks of bias in large language models. *First Monday* (2023). https://doi.org/10.5210/fm.v28i11.13346

[19] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference (ACE '22)*. ACM. https://doi.org/10.1145/3511861.3511863

[20] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference (ACE '23)*. ACM. https://doi.org/10.1145/3576123.3576134

[21] Max Fowler, David H. Smith IV, Mohammed Hassan, Seth Poulsen, Matthew West, and Craig Zilles. 2022. Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study. *Computer Science Education* 32, 3 (2022), 355–383. https://doi.org/10.1080/08993408.2022.2079866

[22] Karin J Gerritsen-van Leeuwenkamp, Desirée Joosten-Ten Brinke, and Liesbeth Kester. 2019. Students' perceptions of assessment quality related to their learning approaches and learning outcomes. *Studies in Educational Evaluation* 63 (2019), 72–82.

[23] Ronald A Goebel and Charles G Stewart. 1971. Effects of experimenter bias and induced subject expectancy on hypnotic susceptibility. *Journal of Personality and Social Psychology* 18, 2 (1971), 263.
[24] Judith TM Gulikers, Theo J Bastiaens, Paul A Kirschner, and Liesbeth Kester. 2006. Relations between student perceptions of assessment authenticity, study approaches and learning outcome. *Studies in Educational Evaluation* 32, 4 (2006), 381–400. https://doi.org/10.1016/j.stueduc.2006.10.003
[25] Mark Guzdial. 2007. Contextualized Computing Education Increasing Retention by Making Computing Relevant. (2007).
[26] Mark Guzdial. 2010. Does contextualized computing education help? *ACM Inroads* 1, 4 (2010), 4–6. https://doi.org/10.1145/1869746.1869747
[27] Kilem Li Gwet. 2008. Computing inter-rater reliability and its variance in the presence of high agreement. *Brit. J. Math. Statist. Psych.* 61, 1 (2008), 29–48.
[28] Kilem L Gwet. 2014. *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters.* Advanced Analytics, LLC.
[29] David Z. Hambrick, Brooke N. Macnamara, and Frederick L. Oswald. 2020. Is the Deliberate Practice View Defensible? A Review of Evidence and Discussion of Issues. *Frontiers in Psychology* 11 (2020). https://doi.org/10.3389/fpsyg.2020.01134
[30] Monica J Harris and Robert Rosenthal. 1985. Mediation of interpersonal expectancy effects: 31 meta-analyses. *Psychological bulletin* 97, 3 (1985), 363.
[31] Mohammad Hassany, Peter Brusilovsky, Jiaze Ke, Kamil Akhuseyinoglu, and Arun Balajiee Lekshmi Narayanan. 2024. Human-AI Co-Creation of Worked Examples for Programming Classes. *arXiv preprint arXiv:2402.16235* (2024).
[32] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the responses of large language models to beginner programmers' help requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1.* 93–105.
[33] Suzanne Hidi and K. Ann Renninger. 2006. The Four-Phase Model of Interest Development. *Educational Psychologist* 41, 2 (2006), 111–127. https://doi.org/10.1207/s15326985ep4102_4
[34] Sigve Høgheim and Rolf Reber. 2015. Supporting interest of middle school students in mathematics through context personalization and example choice. *Contemporary Educational Psychology* 42 (2015), 17–25. https://doi.org/10.1016/j.cedpsych.2015.03.006
[35] Sigve Høgheim and Rolf Reber. 2017. Eliciting Mathematics Interest: New Directions for Context Personalization and Example Choice. *The Journal of Experimental Education* 85, 4 (2017), 597–613. https://doi.org/10.1080/00220973.2016.1268085
[36] Mollie Jordan, Kevin Ly, and Adalbert Gerald Soosai Raj. 2024. Need a Programming Exercise Generated in Your Native Language? ChatGPT's Got Your Back: Automatic Generation of Non-English Programming Exercises Using OpenAI GPT-3.5. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1.* 618–624.
[37] Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *Proceedings of the 26th Australasian Computing Education Conference (ACE 2024).* ACM. https://doi.org/10.1145/3636243.3636252
[38] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the potential of large language models to generate formative programming feedback. In *2023 IEEE Frontiers in Education Conference (FIE).* IEEE, 1–5.
[39] Olivier Klein, Stéphane Doyen, Christophe Leys, Pedro A Magalhães de Saldanha da Gama, Sarah Miller, Laurence Questienne, and Axel Cleeremans. 2012. Low hopes, high expectations: Expectancy effects and the replicability of behavioral experiments. *Perspectives on Psychological Science* 7, 6 (2012), 572–584.
[40] Charles Koutcheme. 2023. *Training Language Models for Programming Feedback Using Automated Repair Tools.* Springer Nature Switzerland, 830–835. https://doi.org/10.1007/978-3-031-36272-9_79
[41] Charles Koutcheme, Sami Sarsa, Juho Leinonen, Arto Hellas, and Paul Denny. 2023. *Automated Program Repair Using Generative Models for Code Infilling.* Springer Nature Switzerland, 798–803. https://doi.org/10.1007/978-3-031-36272-9_74
[42] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2022. Time-on-Task Metrics for Predicting Performance. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1.* 871–877.
[43] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023).* ACM. https://doi.org/10.1145/3587102.3588785
[44] Juho Leinonen, Paul Denny, and Jacqueline Whalley. 2021. Exploring the Effects of Contextualized Problem Descriptions on Problem Solving. In *Proceedings of the 23rd Australasian Computing Education Conference (ACE '21).* ACM. https://doi.org/10.1145/3441636.3442302
[45] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023).* ACM. https://doi.org/10.1145/3545945.3569770
[46] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097. https://doi.org/10.1126/science.abq1158
[47] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23).* ACM. https://doi.org/10.1145/3631802.3631830
[48] Ellie Lovellette, Dennis J Bouvier, and John Matta. 2024. Contextualization, Authenticity, and the Problem Description Effect. *ACM Transactions on Computing Education* (2024). https://doi.org/10.1145/3643864
[49] Ellie Lovellette, John Matta, Dennis Bouvier, and Roger Frye. 2017. Just the Numbers: An Investigation of Contextualization of Problems for Novice Programmers. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17).* ACM. https://doi.org/10.1145/3017680.3017726
[50] Anna Ly, John Edwards, Michael Liut, and Andrew Petersen. 2021. Revisiting Syntax Exercises in CS1. In *Proceedings of the 22st Annual Conference on Information Technology Education (SIGITE '21).* ACM. https://doi.org/10.1145/3450329.3476855
[51] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023).* ACM. https://doi.org/10.1145/3545945.3569785
[52] Lauri Malmi, Judy Sheard, Jane Sinclair, Päivi Kinnunen, and Simon. 2023. Domain-Specific Theories of Teaching Computing: Do they Inform Practice?. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23).* ACM. https://doi.org/10.1145/3631802.3631810
[53] Ference Marton and Roger Säljö. 1976. On qualitative differences in learning: I—Outcome and process. *British Journal of Educational Psychology* 46, 1 (1976), 4–11.
[54] Joseph E. Michaelis and David Weintrop. 2022. Interest Development Theory in Computing Education: A Framework and Toolkit for Researchers and Designers. *ACM Trans. Comput. Educ.* 22, 4 (2022). https://doi.org/10.1145/3487054
[55] Julia Mullen, Chansup Byun, Vijay Gadepally, Siddharth Samsi, Albert Reuther, and Jeremy Kepner. 2017. Learning by doing, High Performance Computing education in the MOOC era. *J. Parallel and Distrib. Comput.* 105 (2017), 105–115. https://doi.org/10.1016/j.jpdc.2017.01.015
[56] Maciej Pankiewicz and Ryan S Baker. 2023. Large Language Models (GPT) for automating feedback on programming assignments. *arXiv preprint arXiv:2307.00150* (2023).
[57] Philipp Peess, Annabell Brocker, Rene Roepke, and Ulrik Schroeder. 2023. A Grammar and Parameterization-Based Generator for Python Programming Exercises. (2023).
[58] Fábio Perez and Ian Ribeiro. 2022. Ignore Previous Prompt: Attack Techniques For Language Models. In *NeurIPS ML Safety Workshop.*
[59] Tung Phung, Victor-Alexandru Pădurean, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors. *arXiv preprint arXiv:2306.17156* (2023).
[60] Philip M Podsakoff, Scott B MacKenzie, Jeong-Yeon Lee, and Nathan P Podsakoff. 2003. Common method biases in behavioral research: a critical review of the literature and recommended remedies. *Journal of applied psychology* 88, 5 (2003), 879.
[61] Ferran Prados, Imma Boada, Josep Soler, and Jordi Poch. 2005. Automatic generation and correction of technical exercises. In *International conference on engineering and computer education: Icece*, Vol. 5.
[62] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE 2023).* ACM. https://doi.org/10.1145/3623762.3633499
[63] Ben Puryear and Gina Sprint. 2022. Github copilot in the classroom: learning to code with AI assistance. *J. Comput. Sci. Coll.* 38, 1 (2022).
[64] Danijel Radošević, Tihomir Orehavocki, and Zlatko Stapić. 2010. Automatic On-line Generation of Student's Exercises in Teaching Programming. In *Central European Conference on Information and Intelligent Systems.* Faculty of Organization and Informatics Varazdin, 87.
[65] Robert Rosenthal and Donald B Rubin. 1978. Interpersonal expectancy effects: The first 345 studies. *Behavioral and Brain Sciences* 1, 3 (1978), 377–386.
[66] Elizabeth Sach. 2012. Teachers and testing: An investigation into teachers' perceptions of formative assessment. *Educational Studies* 38, 3 (2012), 261–276.

[67] Kay Sambell, Liz McDowell, and Sally Brown. 1997. "But is it fair?": An exploratory study of student perceptions of the consequential validity of assessment. *Studies in Educational Evaluation* 23, 4 (1997), 349–371.

[68] Kate Sanders, Jonas Boustedt, Anna Eckerdal, Robert McCartney, and Carol Zander. 2017. Folk Pedagogy: Nobody Doesn't Like Active Learning. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM. https://doi.org/10.1145/3105726.3106192

[69] Eddie Antonio Santos, Prajish Prasad, and Brett A Becker. 2023. Always provide context: The effects of code context on programming error message enhancement. In *Proceedings of the ACM Conference on Global Computing Education Vol 1*. 147–153.

[70] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. 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 2022)*. ACM. https://doi.org/10.1145/3501385.3543957

[71] Sami Sarsa, Arto Hellas, and Juho Leinonen. 2022. Who Continues in a Series of Lifelong Learning Courses?. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Dublin, Ireland) *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 47–53. https://doi.org/10.1145/3502718.3524752

[72] Johanna Schoenherr. 2024. Personalizing real-world problems: Posing own problems increases self-efficacy expectations, intrinsic value, attainment value, and utility value. *British Journal of Educational Psychology* (2024). https://doi.org/10.1111/bjep.12653

[73] Mariana Solari, María Isabel Vizquerra, and Anna Engel. 2022. Students' interests for personalized learning: an analysis guide. *European Journal of Psychology of Education* 38, 3 (2022), 1073–1109. https://doi.org/10.1007/s10212-022-00656-3

[74] Peter Sovietov. 2021. Automatic generation of programming exercises. In *2021 1st International Conference on Technology Enhanced Learning in Higher Education (TELE)*. IEEE, 111–114.

[75] Sandro Speth, Niklas Meißner, and Steffen Becker. 2023. Investigating the Use of AI-Generated Exercises for Beginner and Intermediate Programming Courses: A ChatGPT Case Study. In *2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 142–146.

[76] Katrien Struyven, Filip Dochy, and Steven Janssens. 2003. Students' perceptions about new modes of assessment in higher education: A review. *Optimising new modes of assessment: In search of qualities and standards* (2003), 171–223.

[77] Shelsey Sullivan, Hillary Swanson, and John Edwards. 2021. Student Attitudes Toward Syntax Exercises in CS1. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*. ACM. https://doi.org/10.1145/3408877.3432399

[78] Leonard Tetzlaff, Florian Schmiedek, and Garvin Brod. 2020. Developing Personalized Education: A Dynamic Framework. *Educational Psychology Review* 33, 3 (2020), 863–882. https://doi.org/10.1007/s10648-020-09570-w

[79] Thomas James Tiam-Lee and Kaoru Sumi. 2018. Procedural Generation of Programming Exercises with Guides Based on the Student's Emotion. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 1465–1470. https://doi.org/10.1109/SMC.2018.00255

[80] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. 2023. Is ChatGPT the Ultimate Programming Assistant – How far is it? https://doi.org/10.48550/ARXIV.2304.11938

[81] Mart Van Dinther, Filip Dochy, Mien Segers, and Johan Braeken. 2014. Student perceptions of assessment and student self-efficacy in competence-based education. *Educational Studies* 40, 3 (2014), 330–351.

[82] Kristina von Hausswolff, Anna Eckerdal, and Michael Thuné. 2020. Learning to program hands-on: a controlled study. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research (Koli Calling '20)*. ACM. https://doi.org/10.1145/3428029.3428058

[83] Akiyoshi Wakatani and Toshiyuki Maeda. 2015. Automatic Generation of Programming Exercises for Learning Programming Language. In *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*. 461–465. https://doi.org/10.1109/ICIS.2015.7166637

[84] Akiyoshi Wakatani and Toshiyuki Maeda. 2016. Evaluation of Software Education Using Auto-generated Exercises. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. 732–735. https://doi.org/10.1109/CSE-EUC-DCABES.2016.269

[85] Candace Walkington and Matthew L. Bernacki. 2017. Personalization of Instruction: Design Dimensions and Implications for Cognition. *The Journal of Experimental Education* 86, 1 (2017), 50–68. https://doi.org/10.1080/00220973.2017.1380590

[86] Candace Walkington and Matthew L. Bernacki. 2018. Personalizing Algebra to Students' Individual Interests in an Intelligent Tutoring System: Moderators of Impact. *International Journal of Artificial Intelligence in Education* 29, 1 (2018), 58–88. https://doi.org/10.1007/s40593-018-0168-1

[87] Candace Walkington and Matthew L. Bernacki. 2020. Appraising research on personalized learning: Definitions, theoretical alignment, advancements, and future directions. *Journal of Research on Technology in Education* 52, 3 (2020), 235–252. https://doi.org/10.1080/15391523.2020.1747757

[88] Sierra Wang, John Mitchell, and Chris Piech. 2024. A large scale RCT on effective error messages in CS1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 1395–1401.

[89] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. ACM. https://doi.org/10.1145/3545945.3569830

[90] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J. Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education* 29, 2–3 (2019), 205–253. https://doi.org/10.1080/08993408.2019.1565235

[91] Laura Zavala and Benito Mendoza. 2018. On the Use of Semantic-Based AIG to Automatically Generate Programming Exercises. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 14–19. https://doi.org/10.1145/3159450.3159608

[92] Jun Zhao, Zhihao Zhang, Luhui Gao, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. LLaMA Beyond English: An Empirical Study on Language Capability Transfer. arXiv:2401.01055 [cs.CL]

[93] Edward J Zoble and Richard S Lehman. 1969. Interaction of subject and experimenter expectancy effects in a tone length discrimination task. *Behavioral Science* 14, 5 (1969), 357–363.