Instructor Perceptions of AI Code Generation Tools – A Multi-Institutional Interview Study

Judy Sheard
Monash University
Melbourne, Australia
judy.sheard@monash.edu

Paul Denny
University of Auckland
Auckland, New Zealand
p.denny@auckland.ac.nz

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

Juho Leinonen
University of Auckland
Auckland, New Zealand
juho.leinonen@auckland.ac.nz

Lauri Malmi
Aalto University
Espoo, Finland
lauri.malmi@aalto.fi

Simon
Unaffiliated
Wadalba, Australia
simon.unshod@gmail.com

ABSTRACT

Much of the recent work investigating large language models and AI Code Generation tools in computing education has focused on assessing their capabilities for solving typical programming problems and for generating resources such as code explanations and exercises. If progress is to be made toward the inevitable lasting pedagogical change, there is a need for research that explores the instructor voice, seeking to understand how instructors with a range of experiences plan to adapt. In this paper, we report the results of an interview study involving 12 instructors from Australia, Finland and New Zealand, in which we investigate educators' current practices, concerns, and planned adaptations relating to these tools. Through this empirical study, our goal is to prompt dialogue between researchers and educators to inform new pedagogical strategies in response to the rapidly evolving landscape of AI code generation tools.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

programming education, instructor perceptions, large language models, LLMs, AI code generation, interview study, generative AI

ACM Reference Format:
Judy Sheard, Paul Denny, Arto Hellas, Juho Leinonen, Lauri Malmi, and Simon. 2024. Instructor Perceptions of AI Code Generation Tools – A Multi-Institutional Interview Study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630880

1 INTRODUCTION

The landscape of computing education is evolving rapidly, with large language models (LLMs) powering a new generation of tools such as ChatGPT and GitHub Copilot driving the transformation [9]. The increasing use of these AI applications brings both challenges and opportunities to educators in the field [30]. One widely voiced concern is the capability of the tools to accurately solve homework problems and exam questions, which may lead to student over-reliance [11, 31]. On the other hand, recent work has shown that AI tools can be used to produce high-quality learning resources, including programming exercises [32], explanations of code [24, 28], and even on-demand feedback to students [27].

This fast-paced evolution has sparked discourse on the benefits and challenges of LLMs [17]. Several interview and survey studies have attempted to gauge instructors' perceptions and attitudes toward AI-enabled tools in the broader educational context [2, 5, 16]. However, such studies do little to account for the unique challenges specific to computing education. For example, the concept of code reuse, where educators often encourage students to use libraries or other pre-existing code. This teaching practice can blur the line between code reuse and plagiarism, making it difficult to establish clear guidelines around acceptable practice [1, 33, 34, 36].

To date, there has been very little work investigating the perceptions of computing instructors toward AI code generation tools. We are aware of just two notable, and very recent, exceptions. Lau and Guo [19] interviewed 20 instructors across nine countries, presenting a hypothetical scenario in which AI tools worked perfectly and asking how they planned to adapt. Zastudil et al. [39] interviewed 12 students and six instructors, all from the same institution, investigating experiences and preferences for AI tools in computing classrooms.

In the present study, complementing the findings of Lau and Guo [19] and Zastudil et al. [39], we outline results from an interview study where we interviewed 12 instructors from three countries in order to capture a diverse range of perspectives on AI applications in computing education. To ground our findings in real-world experiences, we asked the instructors about their familiarity with and perceptions of LLMs, and their proposed strategies to aligning assessment practices with the evolution of the field and to tackle concerns of plagiarism.

2 RELATED WORK

This study builds upon a growing body of research on the applications of large language models (LLMs) in computing education, which has received significant attention over the past year.

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03.
https://doi.org/10.1145/3626252.3630880

2.1 Large Language Models in Computing Education Research

A large portion of LLM-related research in computing education has centered on assessing the capability of LLMs for solving programming exercises. One of the first papers on LLMs in computing education looked into how well the Codex model can solve introductory programming exercises [11]. The study found that Codex could solve around 80% of introductory programming (CS1) problems that had been included in past exams, a performance that would put Codex in the top quartile of students in the course. Similar performance was found in a follow-up study that used Codex to solve more complex problems in data structures and algorithms (CS2) [12].

With the subsequent emergence of tools such as GitHub Copilot, which is powered by Codex, researchers have looked into the capabilities of these tools, noting that they are able to solve around half of introductory programming exercises directly, and another 30% with some additional 'prompt engineering', i.e., additional guidance provided to the model as part of the prompt [7]. However, other work has found that Copilot might struggle to follow instructions when used to solve programming exercises [37], and that LLMs can struggle to solve computational thinking tasks [3] and when asked to identify bugs in code, may point out non-existent issues [15].

The emergence of LLM code generation tools such as Copilot has also led to work examining students' interaction with such tools. Interacting with an LLM has been observed to lead to higher performance in code writing tasks but to little differences in learning gains [18]. Novice programmers also sometimes struggle to use Copilot; for example, spending a lot of time unsuccessfully trying to coerce Copilot to produce specific code [31]. One recent suggestion has been to train students to better interact with LLMs using 'prompt problems' [8].

Another stream of research has explored the use of LLMs to create and enhance educational resources such as programming exercises [32], code explanations [24, 28, 32], and programming error messages [25, 29]. LLMs can be used to create novel programming exercises with prescribed themes (such as basketball) and programming concepts (such as loops) [32], as well as line-by-line explanations of code, which could potentially benefit novice programmers [32]. Two follow-up studies found that novices use explanations created by LLMs and generally rate them as being useful for learning [28], and that the code explanations created by LLMs are rated as being better summaries of the code and more understandable than those created by students [24]. LLMs can also be used to enhance programming error messages, researchers suggesting that the enhanced error messages were better than the original ones around half of the time [25]. Recent work has also highlighted the possibility of controlling the comprehensiveness and accuracy of syntax error feedback [29].

2.2 Student and Instructor Perspectives of Large Language Models

Literature looking into student and instructor perspectives of large language models within CER is still scarce, though there are a few notable exceptions. An interview of 20 instructors conducted by Lau and Guo [19] found that in the short term, most instructors intended to implement immediate measures to deter cheating, while the long-term plans were divided. Some instructors were in favor of banning AI tools to maintain a focus on teaching programming fundamentals, whereas others advocated for the integration of AI tools into courses to prepare students for future job markets in which AI would be widely used. The study, however, involved a hypothetical LLM that worked flawlessly, serving as an 'oracle', a level that current LLMs still appear far from attaining.

Zastudil et al. [39] interviewed 12 students and six instructors. Students said that the AI tools reduced the effort of writing code, helped them to find learning materials, and allowed them to avoid busy work. They also voiced concerns about over-reliance on and trustworthiness of the AI tools, and noted that the availability of such tools might increase plagiarism. Instructors saw benefits of the AI tools in providing explanations of code, helping students to find inspiration and to get feedback, and allowing access to high quality learning resources. The instructors, too, expressed concerns about over-reliance and plagiarism, and pointed out issues with the trustworthiness of the systems. Concerns about over-reliance were also expressed in a study exploring student use of Copilot, with Prather et al. [31] noting that students expressed concerns about over-reliance despite feeling more productive when using an LLM.

The present study expands on these prior works by exploring instructors' views on programming education in the era of AI code generation tools.

3 METHODOLOGY

3.1 Research Questions

Our study is based on three research questions.

(1) What are the threats and opportunities for learning programming from AI tools?
(2) What are the threats and opportunities for teaching and assessment of programming from AI tools?
(3) How should programming education be changed when students are using AI tools?

3.2 Approach

Our study employs a qualitative research methodology, specifically a phenomenological approach, with the goal of gaining deep insight into programming instructors' experiences, perceptions, and attitudes toward the use of Language Learning Models (LLMs) such as ChatGPT and Copilot. This approach was chosen to explore the complexities of teaching experiences and the instructors' nuanced views on the integration of advanced programming aids into their pedagogical practices.

3.3 Participant Recruitment and Characteristics

Twelve programming instructors were purposively selected from universities in Australia, Finland and New Zealand to capture a broad range of teaching experiences, programming contexts, and institutional practices. The universities were selected to represent a mix of institutions with varying sizes and geographical locations. All interviewees had considerable experience in teaching programming, with a range of 10 to 50 years. Most participants (9/12) had experience in teaching both introductory and advanced courses,

while one had only taught introductory courses and two had only taught advanced courses (CS2 and above).

3.4 Data Collection

The primary method of data collection was semi-structured interviews lasting 40-60 minutes, providing a balance between gaining consistent data across participants and allowing individual experiences to emerge. The interviews were conducted either in person or via Zoom, based on participant preference and convenience. The interviews were conducted between December 2022 and May in 2023. The interview guide was carefully constructed, starting with general questions about the participants' teaching experience and gradually moving to more specific inquiries about the potential use of LLMs in programming education. The interview guide is available as an online appendix[1].

The interview guide was divided into several sections. First, instructors were asked about their background, including their length of teaching experience, the types and sizes of programming courses they had taught, and the programming languages and supporting technologies they had used. Next, they were asked to reflect on the learning objectives, assessment methods, and their alignment, in their recently taught programming courses. Then they were prompted to share their views on the potential opportunities and threats of using AI tools in their courses, and their implications for student learning and plagiarism. Lastly, they were asked to envision potential revisions in pedagogical practices due to the use of such tools, and to contemplate the future landscape of programming education.

Before the interview, participants were shown a brief video of Copilot to ensure they had a shared understanding of its features and functionality. This was critical to stimulate meaningful reflections and discussions on its potential implications.

3.5 Analysis

All interviews were audio-recorded with the prior consent of the participants and then transcribed verbatim to preserve the richness of the data. The transcriptions were pseudonymized to maintain participant confidentiality.

The data were analyzed following Braun and Clarke's six-step process for thematic analysis [4]. The first author initially read and reread the transcripts, noting initial ideas and generating initial codes. These codes were sorted into potential themes and subthemes, which were reviewed and refined to ensure that they accurately represented the coded extracts. The codes, themes, and subthemes were then reviewed by another author. Any differences were discussed and agreement reached.

The themes were then organized into three groups, each relating to one of the research questions. Eight themes were identified. Two themes *learning* and *student use of tools* related to RQ1, three themes *teaching, assessment* and *academic integrity*, related to RQ2, and three themes *programming education, policy* and *AI tools* related to RQ3.

[1]Available at https://osf.io/2czg6/?view_only=ec32ff862bf34aba9e850de268139289

3.6 Ethical Considerations

This study was conducted in accordance with ethical guidelines established by the respective universities of the researchers and was approved as appropriate by their Institutional Review Boards (IRB). All participants were provided with an informed consent form that detailed the study's purpose, the nature of their participation, and their right to withdraw at any time without penalty. To ensure privacy and confidentiality, data was shared among the researchers only in a pseudonymous format.

4 RESULTS

The 12 interviewees, four female and eight male, came from six institutions in Australia, Finland and New Zealand. The interviewees had a range of experience with AI tools, had seen the tools demonstrated, and were knowledgeable about their capabilities. Four of the interviewees had played with the tools generally and six had explored the capabilities of the tools with their own coursework, for example by generating course content and testing the capabilities of the tools to generate and explain solutions.

We report the findings of the interview data analysis under each of our three research questions. Codes from I1 to I12 were assigned to the 12 interviewees.

4.1 RQ1: Threats and Opportunities for Learning Programming

The interviewees mentioned a number of ways in which AI tools could assist students in their learning of programming but they also expressed many concerns.

A useful feature of the tools is they can be used to readily generate different examples of code. This can be used by students to demonstrate concepts and could motivate learning. The tools can be a source of support for students in addition to their teachers and classmates, providing individual, tailored, and endless explanations.

> *If a student is confused about a concept, they can sit with ChatGPT and it will talk to them for hours about that particular concept.* (I3)

A particular feature of Copilot is that it can give students a starting point if they are stuck, providing a structure and sequence of code. One interviewee likened this to fading worked examples [13], where students can work with some concepts but not have to remember every little detail of the syntax. The fact that generated code is not always correct or appropriate was seen as a positive as this means that students need to read the code and understand it in order to work out how to adapt or fix it, and they would learn from doing this.

A couple of interviewees proposed that Copilot could help students achieve far more than they could on their own, but only if they know how to program. They cautioned that this was not the case for novice programmers, who would not necessarily know what instructions to give to Copilot.

All interviewees expressed concerns about students missing out on learning by using AI tools, a similar finding to the study by Zastudil et al. [39]. A common concern was that students would not go through the crucial thinking processes and steps necessary to learn programming. A student may achieve a correct solution to

a problem through using an AI tool, but then have no idea how to solve a different problem without the tool. Over-reliance on AI tools for solving small problems in introductory programming classes means that students would not learn the fundamental elements of programming and not be prepared for higher level courses with larger tasks that Copilot would not be able to answer. The danger is that students can use the tools and believe they are learning. As one interviewee remarked, *"it is a really great tool to create code but also a really great tool to prevent yourself from learning"* (I9).

The issue of student trust in AI tools was raised by several interviewees who were concerned that students would not always be able to judge the correctness of a solution produced by an AI tool and could be misled or confused by a wrong solution. Such a trust in AI tools could have a negative impact on their learning. A similar issue was raised in the study by Zastudil et al. [39].

A deeper issue was raised about responsibility for learning. By using a tool such as Copilot to write the programming code for their learning tasks, the students are, in effect, using the tool to learn to program for them. Although students may view this as learning *with* Copilot, the students are in effect giving agency to Copilot to learn *for* them. One interviewee saw this as a form of surrogacy, which of course will not work. Learning is the student's own responsibility, although teachers play an important role.

> *It's teachers' responsibility to motivate them and make such a problem that [students] are keen to solve and in a way that they actually would like to learn something and realize that they need these skills also in the future.* (I7)

4.2 RQ2: Threats and Opportunities for Teaching and Assessment

Interviewees suggested various ways the AI tools could assist them with teaching and assessment. They also saw a number of threats from the tools and offered different strategies for addressing these.

Most ideas for the ways AI tools could help with teaching and assessment were related to efficiency in the preparation of learning and assessment tasks. For example, the tools could be used to create a bank of exercises which would provide a greater number and variety of programming tasks than is typically feasible at present. An innovative idea proposed by one interviewee was the creation of personalized tasks that are related to the students' own interests, which could help motivate the students to complete the tasks.

Several interviewees described specific learning tasks that the tools could facilitate. For example, using an AI tool to generate four versions of a simple program using different variable names and asking the students to comment on the comprehensibility of the program code; or using an AI tool to generate faulty solutions to a problem and asking students to identify the problems. This particular example would serve to demonstrate to students the potential problems with AI tools, and would be particularly convincing if done in a live setting.

AI tools could be used to generate solutions to exercises. This can be a way of ensuring that exercises are at the right level of difficulty for the students to solve. This could also assist in the preparation of marking guides for assessment tasks.

There were also suggestions that the AI tools could be incorporated into the teaching programs of higher level programming courses where the students work on projects. By allowing the use of AI tools the students could work alone or in groups to produce larger software systems than currently feasible.

An interesting use of AI tools proposed by one interviewee was an automated system to give feedback to students when they seek help. As the interviewee proposed, *"we ... can give feedback written by a human, or feedback that looks like it was written by a skilled TA, but it is from a large language model"* (I10).

The main threats from AI tools related to teaching were the teachers' lack of preparedness and not knowing how students would use the tools. Several interviewees mentioned the need to suddenly rethink and redesign their courses, as all the current exercises they had for the students could be completed by the tools. The challenge was to find when it would be beneficial to use the tools in the teaching program and when it could cause harm. *"I think part of the answer is in persuading [students] that there really is a point in learning to program for themselves"* (I1).

Assessment was the area where interviewees perceived the most threats, which has also been found in other studies [19, 39]. The main concern was with students using the tools to assist with assessment tasks. If AI tools are allowed in the preparation of an assessment task then it is difficult to assess the level of understanding that the students have achieved through completing the task.

Of more serious concern was the unauthorized use of the tools for assessment tasks, typically take-home assignments. Several interviewees saw this as a form of contract cheating:

> *...there have always been so many ways of cheating, but I don't think I've ever been aware of such an obvious, cheap, and easy way of cheating. Students can get [an AI tool] to answer any question I can ask them at the moment and therefore I have lost my ability to confidently assess any work that students hand in.* (I1)

This form of cheating has a negative impact on the education environment and is unfair to students who spend the time to complete the task as intended but get a comparable mark to the student who used an AI tool.

A challenge is that contract cheating via an AI tool is difficult to detect. A number of approaches were suggested for verifying that the submitted work is that of the student. One interviewee mentioned tools for detecting AI-generated content, but cautioned against their accuracy. A couple of interviewees suggested automated systems to monitor assignment process, either by taking snapshots of the code at different stages of production or by gathering keystroke data. In contrast, several interviewees proposed using oral assessment to verify that the work submitted is that of the student and not of an AI tool.

Some interviewees suggested taking an educative approach by ensuring clear guidelines for the use of AI tools and requiring students to acknowledge any part of their submitted assessment work that was produced with the assistance of an AI tool. This would allow the assessment grading to focus on the code produced by the student and there would be no plagiarism to deal with.

Many interviewees argued the need for some form of invigilated assessment to monitor potential use of the tool, and a number mentioned an exam in a controlled environment. This is despite the

recent trend in many institutions to move from exam assessments to more authentic forms of assessment.

Overall, it was recognized that there is a need to rethink assessment design.

> *...we are clearly living in a time in which we have to completely rethink computing education, and particularly the assessment side of computing education, because we can no longer assess students in any of the ways we have been trying to assess them.* (I1)

The main idea proposed to meet this challenge was using unique and individualized projects, possibly incorporating the use of AI tools. Projects were seen as an authentic way of assessing, and would provide opportunities for students to gain experience using the AI tools in a meaningful setting. Portfolio assessments were also mentioned, but they were viewed by some as problematic as it is now more difficult to tell whether the portfolio is the work of the student.

4.3 RQ3: How Programming Education Should Change

Interviewees agreed that the advent of the AI tools will dramatically change programming education and this was a daunting prospect. As one interviewee remarked, *"it's frightening because it effectively throws away most of the efforts we have gone to in the past to teach people to program."* (I1)

A major challenge facing programming educators is deciding 'what to teach' in programming courses, as what will be required for a programming job is very likely to change in the future. While most interviewees felt it was important that programming professionals had knowledge and practical understanding of the foundations of programming, and that this should be taught in programming courses, some interviewees questioned whether all computing students would need low-level programming skills in the future. An important consideration was how the computing industry will use AI tools.

> *I think we need different kinds of professionals with different understandings of computing. Some need to be very deeply involved with how our programming languages work ... others might only need some kind of overall understanding. They are not programmers by themselves, but they still should understand how software is produced.* (I12)

There was general agreement that it is important to teach students how to interact effectively with AI tools.

> *...if we don't teach the students to use [AI tools] and integrate them, the students will use them ... I'm quite certain that workplaces would, in the future, require the students to use such tools ... So, if we didn't teach them, there would probably be some kind of backlash.* (I9)

A key question is when AI tools should be introduced into computing programs. One interviewee proposed that AI tools should be introduced as early as possible. However, most were concerned that using AI tools in introductory programming courses would interfere with learning the basics of programming. One interviewee expressed a fear that we might move *"toward programming to be more about writing prompts or how to prompt this black box to do something, which is very hard to teach"* (I10).

There was general agreement that the right time to introduce AI tools is once students are proficient with programming. At this stage they could use the tools in projects in a similar way to how they would use them in their future work. The emphasis in teaching would necessarily shift from code writing toward integration of code, code evaluation and testing. One interviewee saw this as part of the continual evolution of programming education with a shift from low-level programming languages to higher levels of abstraction.

Another major challenge was 'how to teach and assess'. There was a general concern that most or all tasks used in CS1 can be readily solved by the AI tools. The risk is that student use of the AI tools could have a negative impact on their learning and could violate academic integrity rules if used for assessment. To avoid situations where students engage in such unproductive behavior, it was proposed that the tasks could be changed to those where the AI tools were not useful; however, with the rapid advances in AI tools, this was seen as difficult to achieve. Tasks used for assessment are particularly problematic.

> *...assuming people say that it's unnatural to code without the tools. Then we need to supervise what kind of things they did in there, whether did they actually understand the code they produced. It changes the learning goals, it changes the learning tasks. It will also possibly change the assessment ... it will change the kind of supervision of the students when they are doing learning.* (I9)

Most interviewees envisaged that changes will be needed to course learning outcomes as the current learning outcomes do not reflect the learning that the students would be gaining through the use of the tool.

> *...assuming that there remains a need to teach people to program, we're going to have to come up with completely different ways of doing it. And with completely different arguments, to persuade students of the value of them, actually learning it rather than them finding a tool to do it. That's an exciting time to be a computing educator.* (I1)

5 DISCUSSION

The whole academic world, including computing faculty members, has been surprised by the power of the new and emerging LLM-based AI tools. The initial focus of discussion among computing departments has been concerns about academic integrity among students, as well as the trivializing of many of the assignments in our introductory courses, which is reflected both in our interviews and in prior studies [19, 39]. However, many teachers also envisage new opportunities to improve education with these tools, providing support for both students and teachers, and this view is also receiving much attention in the current LLM-related research in CER. Overall, we are still at an adaptation phase, and, perhaps because of this, the future of education seems unclear. There are calls to reconsider learning goals at course and curriculum level, and teachers are looking for novel teaching methods and assessment practices that would accommodate the change. Many have some experience in using the tools based on personal exploration, but

there are still no established best practices or research-based pedagogies highlighting when and how the tools should be integrated with different courses.

A curriculum-level perspective poses a fundamental question: who will need programming skills in the future? So far, service courses in programming for non-CS degree programs have often been motivated by gaining a sufficient understanding of programming concepts and process to facilitate collaboration with professional software developers, as well as learning practical skills to write small programs to manipulate their own data. Now, we may be in a phase where such skills are replaced by learning to use AI tools to carry out the necessary programming, in the same way that we use statistical software to perform tests without considering how the calculations are actually carried out. Perhaps only a small number of students need to learn 'real programming skills', implying that computer science would transition from common academic knowledge to an area of deep expertise.

On a course level, many of our interviewees emphasized that assessment should focus more on the process of programming than on the final submitted work, which is what is currently evaluated either manually or automatically in exercises, projects, and exams. It is clear that while formally much of current programming education states that we teach reading, writing, tracing, testing, and debugging programs, most assessment methods focus on the submitted work and its correctness and quality. Naturally, these remain important aspects of assessment and should not be abandoned. The challenge has been that much of students' work during programming is invisible to teachers, and while we can get some snapshots of what is happening in classroom sessions, we see very little of what is happening when students are working on their own. Some work has looked into using fine-grained programming process logs gathered from instrumented IDEs to uncover student work patterns [21–23, 26, 40] and to visualize student programming processes [10, 14, 35, 38]. However, little has been done to explore how AI tools could support the analysis of such extremely rich data to identify situations where students get stuck, tinker with their program, or have clear misconceptions. AI has the potential, based on large sets of log data, to build constructive personal feedback which could be integrated into IDEs. At a simpler level, AI has already been used, for example, to improve error messages [25].

Considering code reading skills, AI tools could be used to generate code examples [6] and code explanations [32]. These could be used as ready-made materials for code review sessions for students, but could also inspire new types of assignment that are automatically generated and possibly automatically assessed using, for example, the 'questions about learner's code' technique [20]. It is easy to imagine AI-driven applications that would generate deliberately buggy code and guide students to find and fix errors, recording their actions and providing personal feedback to support their process.

From the teachers' perspective, AI tools can be used to generate not only examples but complete programming assignments with model solutions and test cases [32], thus reducing their work and supporting the production of personalized exercises for students. The tools could summarize students' code and documents for initial screening, and suggest comments on them which the teacher could accept, reject or modify, thus speeding up the giving of formative feedback.

6 LIMITATIONS

Our study comes with a set of limitations which we outline here. Related to the generalizability of our findings, one limitation is that we interviewed only 12 academics from six different universities. It is possible that the thoughts expressed by these academics do not accurately represent those of the larger population of computing educators. However, even with this number of interviewees, we started seeing similar thoughts being expressed, which could signal that some level of saturation was reached.

Additionally, large language models are advancing very fast, and results published now can be dated in a few months. As an example, a speaker at the ITiCSE 2023 conference started his presentation by apologizing that the results in the paper were unfortunately not longer valid, as a new version of GPT was clearly more powerful than the one used in the paper, making the results dated at best. Similarly, it is likely that the views expressed by the instructors in our study reflect their thoughts on AI with the capabilities that were available at the time of the interviews (early 2023), and their thoughts might differ now that the capabilities have already grown.

7 CONCLUSION

We interviewed 12 teachers from six different institutions in three countries concerning their perspectives on teaching programming in introductory and advanced courses. The interviews revealed concerns about threats to academic integrity and challenges for the learning of novice students if they use the tools to generate submissions and not to learn programming. We acknowledge that similar concerns are shared by most of our colleagues internationally; however, our interview data is rich in discussion of new opportunities and how programming education could or should be revised to mitigate some of the challenges but also to accept that world is now different and we must learn to live in it.

These goals provide fruitful work for computing education researchers, not only in designing and implementing such AI tools but in investigating their impact on students' learning, conceptions, and studying process, as well as the impact on the work of educators. A rich research field is opening for us, along with a vast demand for working solutions and tools, not only in universities but also in school level programming education. Many questions concern how we should teach the appropriate use of these tools at different levels. When is it appropriate? What basic knowledge is needed for using them successfully? How should they be integrated into the curriculum? What skills and knowledge are needed for different target groups? This is a rich field for future work.

ACKNOWLEDGMENTS

We would like to thank the academics who generously gave their time to be interviewed for this study. We are grateful for the grant from the Ulla Tuominen Foundation to Juho Leinonen.

REFERENCES

[1] Ibrahim Albluwi. 2019. Plagiarism in programming assessments: a systematic review. *ACM Trans. Comput. Educ.* 20, 1, Article 6 (Dec 2019), 28 pages. https://doi.org/10.1145/3371156

[2] Sara Amani, Lance White, Trini Balart, Laksha Arora, Dr. Kristi J. Shryock, Dr. Kelly Brumbelow, and Dr. Karan L. Watson. 2023. Generative AI perceptions: a survey to measure the perceptions of faculty, staff, and students on generative AI tools in academia. arXiv:2304.14415 [cs.HC]

[3] Carlo Bellettini, Michael Lodi, Violetta Lonati, Mattia Monga, and Anna Morpurgo. 2023. DaVinci goes to Bebras: a study on the problem solving ability of GPT-3. In *15th International Conference on Computer Supported Education. 2: CSEDU*. SciTePress, 59–69.

[4] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101.

[5] Cecilia Ka Yuk Chan and Katherine K. W. Lee. 2023. The AI generation gap: are gen Z students more interested in adopting generative AI such as ChatGPT in teaching and learning than their gen X and millennial generation teachers? arXiv:2305.02878 [cs.CY]

[6] Paul Denny, Hassan Khosravi, Arto Hellas, Juho Leinonen, and Sami Sarsa. 2023. Can We Trust AI-Generated Educational Content? Comparative Analysis of Human and AI-Generated Learning Resources. arXiv:2306.10509 [cs.HC]

[7] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: exploring prompt engineering for solving CS1 problems using natural language. In *54th ACM Technical Symposium on Computer Science Education V.1*. 1136–1142.

[8] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2023. Promptly: using prompt problems to teach learners how to effectively utilize AI code generators. arXiv:2307.16364 [cs.HC]

[9] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2023. Computing education in the era of generative AI. arXiv:2306.02608 [cs.CY]

[10] Joseph Ditton, Hillary Swanson, and John Edwards. 2021. External imagery in computer programming. In *52nd ACM Technical Symposium on Computer Science Education*. 1226–1231.

[11] 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 *24th Australasian Computing Education Conference*. 10–19.

[12] 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 *25th Australasian Computing Education Conference*. 97–104.

[13] Simon Gray, Caroline St. Clair, Richard James, and Jerry Mead. 2007. Suggestions for graduated exposure to programming concepts using fading worked examples. In *Third International Workshop on Computing Education Research (ICER '07)*. 99–110. https://doi.org/10.1145/1288580.1288594

[14] Kenny Heinonen, Kasper Hirvikoski, Matti Luukkainen, and Arto Vihavainen. 2014. Using codebrowser to seek differences between novice programmers. In *45th ACM Technical Symposium on Computer Science Education*. 229–234.

[15] 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. *arXiv preprint arXiv:2306.05715* (2023).

[16] Jaeho Jeon and Seongyong Lee. 2023. Large language models in education: a focus on the complementary relationship between human teachers and ChatGPT. *Education and Information Technologies* (2023), 1–20.

[17] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences* 103 (2023), 102274.

[18] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI code generators on supporting novice learners in introductory programming. In *2023 CHI Conference on Human Factors in Computing Systems*. 1–23.

[19] Sam Lau and Philip J Guo. 2023. From "ban it till we wnderstand it" to "resistance is futile": how university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot. In *ICER 2023*.

[20] Teemu Lehtinen, Lassi Haaranen, and Juho Leinonen. 2023. Automated questionnaires about students' JavaScript programs: towards gauging novice programming processes. In *25th Australasian Computing Education Conference*. 49–58.

[21] Juho Leinonen. 2019. *Keystroke Data in Programming Courses*. Ph. D. Dissertation. University of Helsinki.

[22] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2021. Does the early bird catch the worm? Earliness of students' work and its relationship with course outcomes. In *26th ACM Conference on Innovation and Technology in Computer Science Education V.1*. 373–379.

[23] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2022. Time-on-task metrics for predicting performance. In *53rd ACM Technical Symposium on Computer Science Education-Volume 1*. 871–877.

[24] 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 *28th Conference on Innovation and Technology in Computer Science Education V.1 (ITiCSE 2023)*. 124–130. https://doi.org/10.1145/3587102.3588785

[25] 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 *54th ACM Technical Symposium on Computer Science Education V. 1*. 563–569.

[26] Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. 2016. Automatic inference of programming performance and experience from typing patterns. In *47th ACM Technical Symposium on Computing Science Education*. 132–137.

[27] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: using large language models with guardrails for scalable support in programming classes. arXiv:2308.06921 [cs.CY]

[28] 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 *54th ACM Technical Symposium on Computer Science Education V.1*. 931–937.

[29] Tung Phung, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generating high-precision feedback for programming syntax errors using large language models. *arXiv preprint arXiv:2302.04662* (2023).

[30] 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. *arXiv preprint arXiv:2310.00658* (2023).

[31] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's weird that it knows what I want": usability and interactions with Copilot for novice programmers. *arXiv preprint arXiv:2304.02491* (2023).

[32] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[33] Judy Sheard and Martin Dick. 2011. Computing student practices of cheating and plagiarism: a decade of change. In *16th Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)*. 233–237. https://doi.org/10.1145/1999747.1999813

[34] Judy Sheard, Simon, Matthew Butler, Katrina Falkner, Michael Morgan, and Amali Weerasinghe. 2017. Strategies for maintaining academic integrity in first-year computing courses. In *2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. 244–249. https://doi.org/10.1145/3059009.3059064

[35] Raj Shrestha, Juho Leinonen, Arto Hellas, Petri Ihantola, and John Edwards. 2022. Codeprocess charts: visualizing the process of writing code. In *24th Australasian Computing Education Conference*. 46–55.

[36] Simon, Judy Sheard, Michael Morgan, Andrew Petersen, Amber Settle, Jane Sinclair, Gerry Cross, and Charles Riedesel. 2016. Negotiating the Maze of Academic Integrity in Computing Education. In *Proceedings of the 2016 ITiCSE Working Group Reports* (Arequipa, Peru) *(ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 57–80. https://doi.org/10.1145/3024906.3024910

[37] Michel Wermelinger. 2023. Using GitHub Copilot to solve simple programming problems. In *54th ACM Technical Symposium on Computer Science Education V.1*. 172–178.

[38] Benjamin Xie, Jared Ordona Lim, Paul K.D. Pham, Min Li, and Amy J. Ko. 2023. Developing novice programmers' self-regulation skills with code replays. In *2023 ACM Conference on International Computing Education Research V.1 (ICER 2023)*. https://doi.org/10.1145/3568813.3600127 second and third authors made equal contributions.

[39] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in computing education: perspectives of students and instructors. arXiv:2308.04309 [cs.HC]

[40] Albina Zavgorodniaia, Raj Shrestha, Juho Leinonen, Arto Hellas, and John Edwards. 2021. Morning or evening? An examination of circadian rhythms of CS1 students. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET).