LLM Practice Problem Generation in a Beginning Programming Class

Kevin Gold

Boston University, Boston, MA 02215, USA
klgold@bu.edu

Abstract. We describe a programming problem generator powered by OpenAI's GPT-4o (or more recently, o1) that was used by an introductory programming class to prepare for midterms. The system had 59 unique student users out of a class of 192. We found that the system was much better at producing problems with solutions that were in the scope of the course material when it used o1 as the underlying LLM instead of GPT-4o; the model change raised the percentage of in-scope problems from 47.5% to 85%. The students generated rather more problems than expected, with an average of 16 or 17 problems per student per exam. However, there was no evidence that student grades on their midterms improved as a result of interacting with the problem generator.

Keywords: Large language models (LLMs) · Generative AI · ChatGPT · Educational technology · Higher education · Programming education

1 Introduction

Not every student wants additional practice problems before the midterm, but many do. Luckily, large language models (LLMs) offer the promise of unlimited practice problems. But students asking for such problems on their own face a challenge: how can they engineer the prompt so that the problems are within the scope of the current course? Students trying to query LLMs on their own for such problems might face the frustration of getting a problem that was unsolvable using the techniques covered in their own course. Students might also ask for problems that are about material the instructor considers unimportant, or get such problems from the AI. An LLM trying to generate problems may lack the context to make a good problem for the course, even if the problem is ostensibly a good problem in the abstract.

We have created a GPT-based practice problem generator for use in an introductory data science course that teaches programming. The practice problem generator specifically makes programming practice problems that touch on topics that the student selects with a "tickybox" interface. To keep the problems relevant to the course, the LLM is provided in the prompt with all the programming examples covered in the course, and it is instructed to only use programming keywords and techniques that are illustrated in that code. We experimented with the ability of two models to follow that directive, and found that while GPT-4o

was able to stay within the bounds of the course some of the time, o1 was able to stay within bounds most of the time - almost twice as often.

We also logged the students' use of the system and were thereby able to answer questions like, how many queries did a typical student make? How many unique student users did the system have? How many topics were in a typical query? And finally, could we perceive the impact on student grades? Selection effects make it difficult to definitively answer the last question, and we do not know if there was any effect of more motivated students using the system more, or perhaps struggling students using the system more. But we can report what we observed, including the size of the effect when students who hadn't been using the system for midterm 1 (an exam halfway through the semester), started using it for midterm 2 (an exam four fifths of the way through the semester).

Section 2 will explore some related work in this space. Section 3.1 will detail the design of the AI tutor. Section 3.2 will describe the class that the tutor assisted with. Section 4 will report our observations of the system and the students. We then will offer some conclusions and next steps in Section 5.

2 Related Work

Experimenting with LLM-generated practice problems has immediate precedents in [8], [15], and [12]. [8] explored the effects of letting students choose their own themes for problems, and found that students generally responded positively to this intervention. [15] used OpenAI Codex to generate problems, and found that 75% of problems were judged to be sensible, 82% were judged novel, 79% matched the requested topic, and only 30% passed their own suite of tests. [12] looked at how LLMs could use existing programming puzzles to make more complex, novel, and challenging problems. However, none of these methods took place in the context of a course where students were trying to use the problem generator to help study for an exam, and thus none explored the issue of which LLM is best for generating problems scoped to the course, nor did they study the impact of using the system on grades; these are our key contributions.

Large language models like ChatGPT have had mixed success in evaluation for academic purposes [7][13]. Academic gains were shown by students who were given 10 days of access to ChatGPT where they were encouraged to ask questions about the subject matter [20]. [2] did not measure its agents' effect on student performance, but their system performed well on the researchers' measures of accuracy, fluency, empathy, engagement, and relevance. [11] used ChatGPT to provide hints to students, but found a 30% error rate that did not compare favorably to a human. [4] assessed the use of ChatGPT to judge student writing; the system had a clear bias toward positive sentiment over the instructor, and the ultimate effects on student performance were not assessed.

Our work also lives in the same space as intelligent tutoring systems (ITSes), computer-assisted instruction (CAI), and human tutors, as all of these are trying to improve the student's academic success through guided practice. For ITSes, which are historically expert systems, [6] reports an average gain in performance

of 0.66 standard deviations. [5] found that the gains from ITSes might actually come from freeing the instructor to give personalized instruction, rather than the benefit coming from working the problems per se. Older computer-assisted instruction (CAI) systems seem to raise scores by about 0.3 standard deviations [18]. Human tutors' effect on student grades is unclear, with estimates ranging from 0.4 [6] to 2 [18]. Compared to these systems, simply providing practice problems is rather more hands-off, as the system doesn't provide any kind of step-by-step working of the problems or feedback. But it does provide the promise of quickly adapting to the needs of the particular student and course.

3 Methods

3.1 The Problem Generator

The students interacted with the GUI shown in Figures 1 and 2. The topics the students could select included "if", "lists", "while", "boolean operators," "for loops," "nested loops," "functions", "sets or dictionaries", "matplotlib", "numpy", "pandas", "regular expressions", "files and exceptions", "object-oriented python", "recursion", and "graphs". Multiple topics could be selected.

The student's selection was passed on to GPT-4o with the following instructions:

> Give me a practice problem for an introductory course in python and data science that uses the following concepts: [list of concepts] The first section of your response should say "PROBLEM" followed by the problem. The second section of your response should say "SOLUTION" before the Python code that solves the problem. The solution must be simple, because this is practice for an exam where students will have limited time. The third section of your response should say "HINT" before a hint that would help with the one issue the student was most likely to get stuck on. Additionally, all Python keywords, syntax, and concepts that you use in the solution must be drawn from the following code from lecture: [lecture code]

The lecture code was all the python code that had appeared somewhere in lecture for the course. (This restriction was tried for the second midterm, late in the course, but wasn't yet implemented before the first midterm, which was halfway through the course.) This all fit into the context window with no need for additional compression or prompt engineering.

The response was then broken up into an initial problem to show the student, a hint that was only shown if the student clicked on a hint request button, and a solution that was only shown if the student clicked on the "show solution" button.

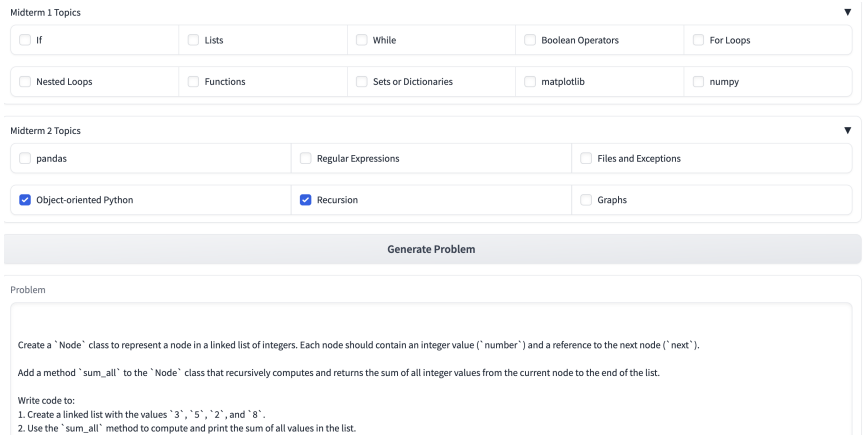

Fig. 1. Top half of the GUI, used by the students to request practice problems.

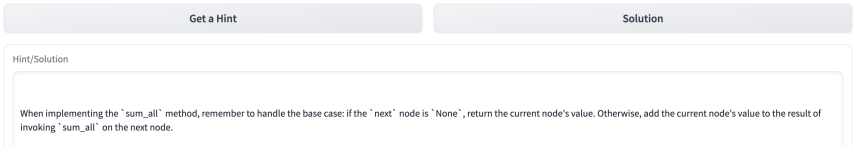

Fig. 2. Bottom half of the GUI, used by the students to request hints and solutions.

3.2 The Course

The problem generator was deployed in a course that introduced students to programming through Python, as well as covering other data science topics. 192 students were enrolled. The course taught how to program in Python over the first half of the course, then covered assorted topics such as running time and graphs in the second half of the course.

The first midterm, delivered close to halfway through the course, concentrated on coding in Python, with multiple choice questions such as "What output is expected from the following code?" and "Which line should be corrected to make this code function?" in addition to two paper-and-pencil programming questions. The second midterm, delivered four-fifths of the way through the course, contained review questions about programming, multiple choice questions about other data science topics (proper machine learning practices, visualization, etc), and another two paper-and-pencil free-response programming questions. Both midterms were wholly in-class and "paper-and-pencil," and served as insurance that students could not get good grades by just relying heavily on generative AI.

Weekly homework was assigned that was distinct from the practice problem generator problems. The homework throughout the course was delivered via Jupyter notebooks, where students would usually write code to answer a

prompt ("Write a function that ..."), but they might occasionally need to answer a thought question (such as needing to analyze the running time of some code, or explain that they are observing the phenomenon of overfitting). A difference from the practice problems is that these problems often had a story or situation associated with them and multiple parts to the problem that developed it. Homework was for credit, while the practice problems were not. Both served as practice that prepared the student for the midterm exams' paper-and-pencil programming.

The topics covered by the course included: introductory variables, branching, and other essentials; introduction to matplotlib, numpy, and loops; functions and dictionaries; dataframes; string manipulation; files; object-oriented programming; recursion; machine learning with scikit-learn; SQL; advanced pandas; and complexity.

For help, two instructors, three graduate teaching assistants, and two undergraduate course assistants each offered three hours a week of office hours. (None of this staff besides the author interacted with or maintained the problem generator.) The course met three times a week. The textbook, Deitel and Deitel's Introduction to Python for Computer Science and Data Science, was recommended to students as an additional source of practice problems, but its structure only loosely matched the structure of the course, and the book was optional.

No direct credit incentive was given for doing problems created by the generative AI, which was offered to students as a way to practice for the midterms.

4 Results

RQ1: What proportion of students would use an optional practice problem generator, and how much?

Out of a class of 192, 27 unique students used the system before the first midterm, and 59 unique students used it between the first and second midterm.

The average number of problems generated per student was 16 leading up to the first midterm, and 17 between the first and second midterm. The largest number generated for a single student for each exam was 76 for the first midterm and 89 for the second. (To put these numbers in perspective, a single homework assignment would typically have roughly 7 problems of the same scope as these generated problems.) Figures 3 and 4 show the distributions of number of queries per student.

RQ2: How did the students make use of the ability to request specific subject matter in their problems?

Figure 5 shows the distribution on the number of topics requested in a single query. It seems students most often (1135 queries) requested just a single topic in their problems, meaning most of the queries could have been handled by, for example, a single dropdown menu. But 608 other queries requested two or more topics in the problem - most commonly, two (211 queries). We had thought

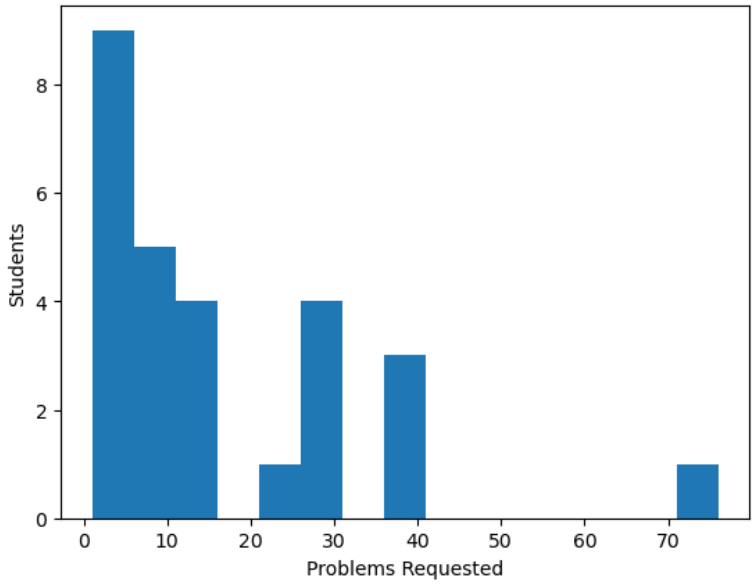

Fig. 3. Distribution of the number of queries per student leading up to midterm 1.

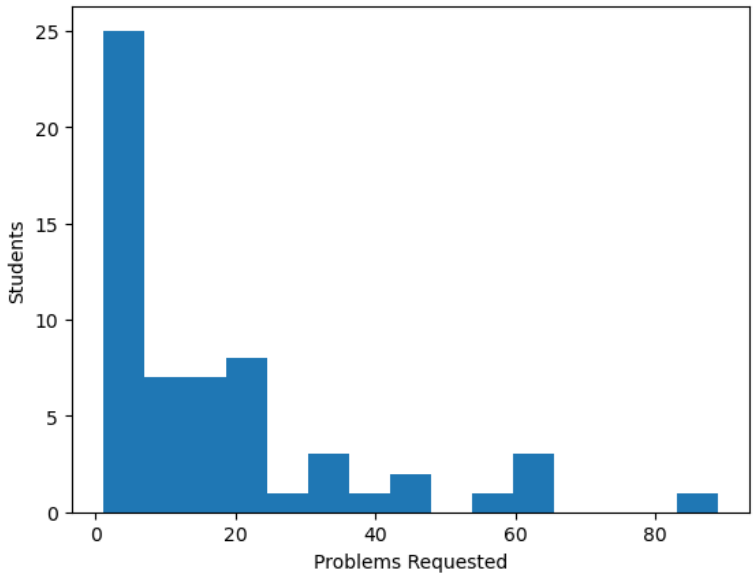

Fig. 4. Distribution of the number of queries per student leading up to midterm 2.

that more students would check all the boxes, but this was not a very common strategy.

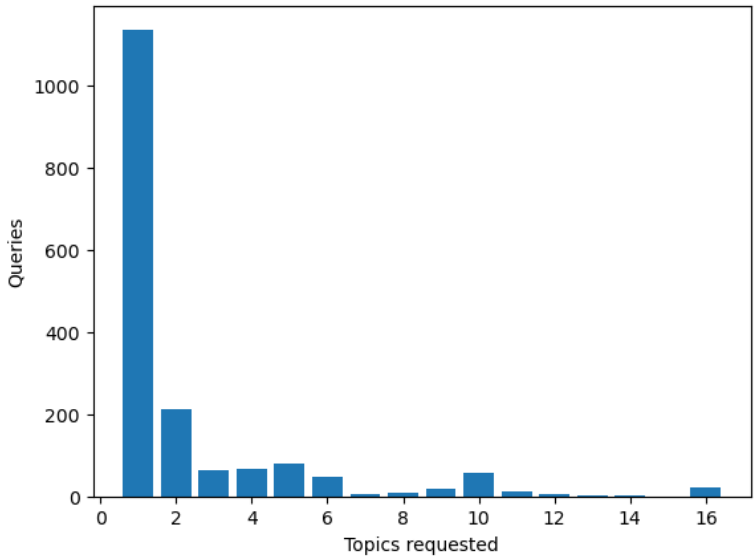

Fig. 5. Distribution of the number of topics requested in each query, across both midterms.

The most common subjects requested were object-oriented programming (416 queries), recursion (410 queries), and pandas (400 queries). These were all subjects that students found difficult, but that also had solutions that tended to stay "in-bounds" of the course coverage more than others using GPT-4o.

It's worth noting that one advantage of the "tickybox" system is allowing the instructor to know what subjects the students feel they could use more practice on, for use in homework later (Figure 6). Extracting the subject matter of freeform queries to an LLM could take more work and be error-prone.

RQ3: Did the problems stay "in bounds" for the material of the course?

40 problems were sampled from the logs for the version of the system that instructed GPT-4o to keep the problems within the bounds of the Python covered in lecture. Of these, it was found only 47.5% of the problems stayed "within bounds" for their solutions, with the rest introducing material that was not covered in the course. (The in-bounds determination was made by the present author, who wrote the course materials and could easily tell at a glance when material fell outside the scope of the course's Python coverage.) Most problems were solvable in some way using tools for beginners, but it was just the solution generated by the AI that used techniques that were too advanced, including dic-

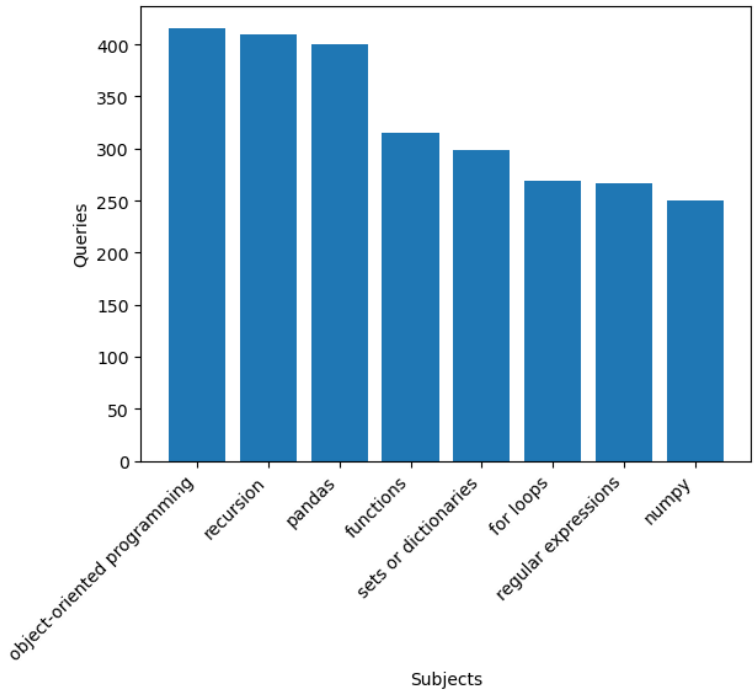

Fig. 6. Number of queries for the 8 most popular subjects.

tionary comprehensions, the use of lambda functions, and other material that wasn't covered in the course yet. (Experiments with 20 samples and no scope-limiting instructions suggest this performance is roughly no better than if there were no scope-limiting instruction at all.)

The second version of the system, which was deployed in a later semester, used the OpenAI o1 model instead of GPT-4o, and this helped tremendously, bumping the percent of queries that were "in-bounds" to 85% in a sample of 40 student queries that were rerun with the new system. It seems the instruction to stay within the bounds of the Python discussed in lecture was too complex for GPT-4o to solve, but not o1. The difference in frequency of staying in-bounds was statistically significant ($p < 0.001$) relative to the base model, using a Chi-squared test.

Figure 7 illustrates the difference in percentages of questions with answers that were in-bounds.

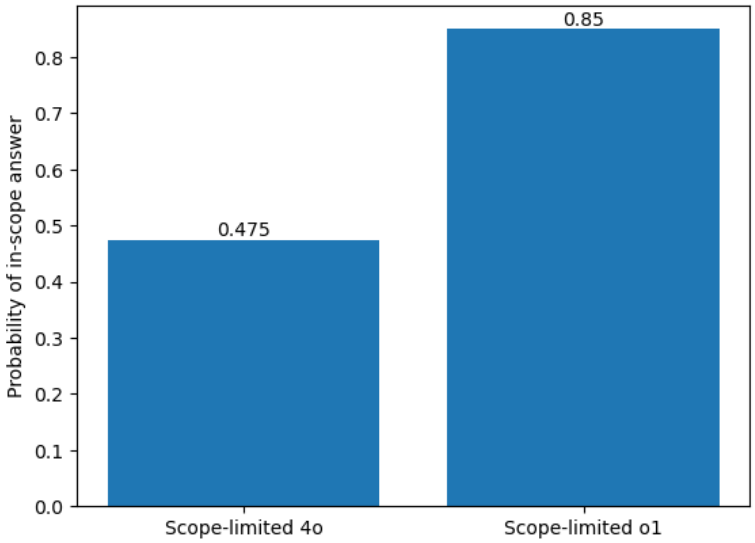

Fig. 7. Different systems' ability to produce problems within the scope of the course.

RQ4: Did using the system appear to improve student grades?

This is a somewhat difficult question to answer, because there may be a certain level of student who is attracted to the system - diligent enough to want to do extra work, but not experienced enough to feel comfortable with their preparation. There could therefore be a selection effect in who uses the system. Nevertheless, for the first midterm, we performed a t-test on the midterm grades of students who used the system versus those who did not, and we found no difference ($p = 0.89$). So if there was a selection effect for students who were doing worse in the course, it was not stronger than the benefit of using the system. On

the second midterm, students seemed to do 4% better if they used the practice problems, but this difference only approached significance ($p < 0.165$)

To try to get at causality, we next looked at the grades of students who didn't use the system for the first midterm, but did use the system for the second midterm, to see if their grades improved. The thought was that if the system is effective, these grades should improve; and it would be less likely to be influenced by selection effects because if certain students were predisposed to using the system, they would have already signed up in the first round. Students who were new users saw a bump in midterm scores 1% higher than other students when moving from one midterm to the next, but this difference was not significant ($p = 0.67$).

5 Discussion

We have two major contributions here. The first is a technical contribution - in such a problem generator, it seems that one absolutely needs to use an o1 quality model or similar reasoner, or else the instructions to stay within the bounds of particular course material will be ignored. This difference was stark and surprising, and should be of interest to anyone wishing to create a similar LLM-driven practice problem generator.

But the second finding is that giving students this endless stream of practice problems is not as effective as we might have thought, as the improvement in midterm scores for students who used the system only approached significance. Just giving students more practice is seemingly not all that they need to succeed on exams. We don't actually know that they did the practice problems, which could be one reason why their performance is not what we expected; perhaps they generated a few, none struck their fancy, and they moved on. Perhaps students just generated problems, almost immediately looked at their answers, and said to themselves, "I probably could have done that." Or perhaps the problems generated that were out-of-scope for the class distracted students from the subject matter they should have been studying. We don't have a clear idea as to why the basic idea of giving more practice wasn't more effective. But this is surely one example of AI not really solving a problem on its own - we need to think harder about how this fits into the students' study environment and practices.

In short, the promise of LLM-generated content for courses needs to be tempered by two realizations. One is that a plan for content generation needs a powerful enough LLM to carry it out, or else instructions may be ignored. Another is that throwing problems over the wall at students is likely only going to be so effective, and there probably needs to be some plan in place to ensure they actually do the work, and ideally learn from it.

What would such a plan look like? Offering extra credit for doing these problems is problematic because students could simply use LLMs to solve the same problems they generated. A different move might be to simply change the interface so that they can submit answers to the problem, then get some kind of gamified congratulations for submitting successful solutions. The issue may be

framing student expectations about what they ought to do with these problems - not merely look at them, or even study them, but do them. This is less obvious in a system that can't take and grade solutions.

But it remains possible that this problem generator using o1-generated problems may actually increase scores after all, since the students only ever interacted here with the more problematic system that stayed in bounds half the time (the o1 powered system was developed after the semester in question, but was tested with the students' actual queries of the earlier system). In the future, we will combine o1 problem generation with an ability to submit answers, and possibly remove the ability to see an answer directly altogether, since the presence of the solution button may be tempting students to jump to the solutions without solving them. The exact way in which these problems are presented may have a big effect on how they are used.

This work can be seen as a novel take on ITS systems that offer practice for students outside their normal homework. Exactly how such systems are deployed can make a big difference in the size of the students' academic gains[5]. Still, it is common for some students to request practice problems before the midterms, and this system offered an unlimited supply.

References

1. Baker, R., Corbett, A., Koedinger, K., Wagner, A.: Off-task behavior in the cognitive tutor classroom: when students "game the system." In: Proceedings of CHI 2004: Computer-Human Interaction, pp. 383—390. ACM, New York (2004)
2. Cao, C., Ding, Z., Lin, J., Hopfgartner, F.: AI chatbots as multi-role pedagogical agents: Transforming engagement in CS education. arXiv:2308.03992 (2023). https://arxiv.org/abs/2308.03992
3. Cunningham, S.: Causal Inference: The Mixtape. Yale UP, New Haven and London (2021). https://mixtape.scunning.com/
4. Dai, W., Lin, J., Jin, F., Li, T., Tsai, Y., Gasevic, D., Chen, G.: Can Large Language Models Provide Feedback to Students? A Case Study on ChatGPT. (2023). https://doi.org/10.35542/osf.io/hcgzj
5. Koedinger, K.R., Anderson, J.R.: Effective use of intelligent software in high school math classrooms. Artificial intelligence in education: Proceedings of the world conference on AI in education, pp. 241–248. Association for the Advancement of Computing in Education, Charlottesville, VA (1993)
6. Kulik, J. A., Fletcher, J. D.: Effectiveness of Intelligent Tutoring Systems: A Meta-Analytic Review. Review of Educational Research, **86**(1), pp. 42–78 (2016). https://doi.org/10.3102/0034654315581420
7. Lo, C.K.: What Is the Impact of ChatGPT on Education? A Rapid Review of the Literature. Educ. Sci., **13**, 410 (2023). https://doi.org/10.3390/educsci13040410
8. Logacheva, E., Hellas, A., Prather, A., Sarsa, S., Leinonen, J.: Evaluating Contextually Personalized Programming Exercises Created with Generative AI. Proceedings of the ACM Conference on International Computing Education Research (ICER) (2024). https://arxiv.org/abs/2407.11994
9. Lowe T.: Debugging: The key to unlocking the mind of a novice programmer? In: 2019 IEEE Frontiers in Education Conference (FIE), pp. 1–9. IEEE, New York (2019)

10. Mathews, M., Mitrović, T., Thomson, D.: Analysing High-Level Help-Seeking Behaviour in ITSs. In: Nejdl, W., Kay, J., Pu, P., Herder, E. (eds) Adaptive Hypermedia and Adaptive Web-Based Systems. Lecture Notes in Computer Science, vol. 5149. Springer, Berlin, Heidelberg. (2008). https://doi.org/10.1007/978-3-540-70987-9_42
11. Pardos, Z., Bhandari, S.: Learning gain differences between ChatGPT and human tutor generated algebra hints. arXiv: 2302.06871 (2023). https://doi.org/10.48550/arXiv.2302.06871
12. Pourcel, J., Colas, C., Molinaro, G., Oudeyer, P., Teodorescu, L.: Generating a diversity of challenging programming puzzles with autotelic generative models. Proceedings of NeurIPS (2024). https://arxiv.org/abs/2310.10692
13. Prather, J., Denny, P., Leinonen, J., Becker, B., Albluwi, I., Craig, M., Keuning, H., Kiesler, N., Kohn, T., Luxton-Reilly, A., MacNeil, S., Petersen, A., Pettit, R., Reeves, B., Savelka, J.. The Robots are Here: Navigating the Generative AI Revolution in Computing Education. In: Proceedings of ITiCSE 2023. https://arxiv.org/abs/2310.00658
14. Roll, I., Baker, R., Aleven, V., Koedinger, K.R.: On the benefits of seeking (and avoiding) help in online problem-solving environments. J. Learn. Sci. **23**(4), pp. 537–560 (2014). https://doi.org/10.1080/10508406.2014.883977
15. Sarsa, S., Denny, P., Hellas, R., Leinonen, J.: Automatic generation of programming exercises and code explanations with large language models. Proceedings of the ACM Conference on International Computing Education Research (ICER) (2022). https://arxiv.org/abs/2206.11861
16. Steenbergen-Hu, S., Cooper, H.: A meta-analysis of the effectiveness of intelligent tutoring systems on college students' academic learning. Journal of Educational Psychology, **106**(2), pp. 331–347 (2014). https://doi.org/10.1037/a0034752
17. van Stee, E., Heath, T., Baker, R., Andres, J.M.A., Ocumpaugh, J.: Help seekers vs. help accepters: Understanding student engagement with a mentor agent. In: Proceedings of AIED 2023. pp. 139–150 (2023)
18. VanLehn, K.: The behavior of tutoring systems. International Journal of Artificial Intelligence in Education, **16**, pp. 227–265 (2006)
19. Weidlich, J., Gašević, D., Drachsler, H.: Causal inference and bias in learning analytics: A primer on pitfalls using directed acyclic graphs. Journal of Learning Analytics **9**(3), pp. 183–199 (2022). https://doi.org/10.18608/jla.2022.7577
20. Wu, T.-T., Lee, H.-Y., Li, P.-H., Huang, C.-N., and Huang, Y.-M.: Promoting Self-Regulation Progress and Knowledge Construction in Blended Learning via ChatGPT-Based Learning Aid. Journal of Educational Computing Research, **61**(8), pp. 3–31 (2023). https://doi.org/10.1177/07356331231191125