Decoding AI Impact: Longitudinal Analysis of Code Submissions in Programming MOOCs

Leo Leppänen
University of Helsinki
Helsinki, Finland
leo.leppanen@helsinki.fi

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

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

Erkki Kaila
University of Helsinki
Helsinki, Finland
erkki.kaila@helsinki.fi

Linda Mannila
University of Helsinki
Helsinki, Finland
linda.mannila@helsinki.fi

***Abstract*—This full research paper investigates how the emergence of large language model (LLM)-based tools has influenced student code submissions in Massive Open Online Courses (MOOCs) on programming. The rise of LLMs has introduced new opportunities and challenges in programming education, particularly regarding how students approach programming tasks. As code submissions can be used as artifacts representing the learning process, analyzing submissions before and after the wide-spread adoption of LLMs can provide insight into changes in how and whether students' approaches to solving programming tasks have changed and more broadly whether the way how programming is practiced has evolved.**

Using programming assignment submission data from two University-level programming MOOCs, we explore whether and to what extent code submissions have changed after the emergence of LLMs. Our analyses focus on quantitative metrics from the MOOCs, including number of submissions, code lengths, stylistic aspects, and code complexity and originality. Our results highlight that there are some differences in the high-level metrics extracted from submissions between the pre- and post-LLM era, although the differences are largely subtle. We discuss possible reasons for the subtle differences.

***Index Terms*—Academic support, Learning technology**

I. Introduction

The public release of ChatGPT in late November 2022 launched a flurry of discussion in educational contexts. While providing ample opportunity for learning, the risks were also quickly acknowledged, highlighting the dual nature of large language models (LLMs) in education [1]. Opinions on how to view student usage of the increasingly available and capable LLM-based tools have varied from calls to reconfigure teaching around said tools [2], [3], to calls of their bans from education context [4]. Educators have expressed concerns of students becoming overly reliant on LLMs [5]–[7] and risks related to academic integrity [8], [9], while others highlight the tools' fragile nature and tendency to produce factually incorrect content (also known as 'hallucination') [10].

Despite the plethora of concerns, previous work has indicated that the use of LLMs is abundant in academia [11], [12], and among students [11], [13]. For example, a recent investigation into essays submitted to a MOOC course on the ethics of artificial intelligence indicated student essay lengths and certain stylistic measures had changed drastically since the release of ChatGPT [14]. Since many institutes of higher education view the use of LLMs as academic dishonesty, hard numbers on their use are hard to come by. At the same time, we have observed cases of students asking in peer-support chats whether they e.g. "should be worried" they find it necessary to "ask ChatGPT for help" in almost every assignment.

In this work, we investigate whether we can identify indicators of student use of LLMs in the context of two freely available introductory university-level programming MOOCs. One of the courses uses the Java programming language, while the other uses Python. More specifically, our research questions are as follows:

- **RQ1**: How has the number of submissions and submission code lengths changed over time in the courses?
- **RQ2**: How has the stylistics of code submissions changed over time?
- **RQ3**: How has the complexity and originality of exercise submissions changed over time?

The rest of this work is structured as follows. First, Section II describes some of the key results motivating and relating to the present endeavor. Then, Sections III and IV describe our methodology—including the two courses in the context of which this work is undertaken—as well as the results we obtained. Finally, in Sections V and VI we discuss our results more broadly, contrasting them to previous works, consider the limitations of the present study, discuss potential future work, and draw some final conclusions.

II. Background

A. LLMs and programming

The publication of generally usable LLMs in late 2022 shook the whole education sector. The opportunities were also quickly discovered in the context of programming education [6], [11]: LLMs can be used to generate educational resources, such as example code [15]–[18] and programming exercises [19]–[21] quickly and effortlessly. Moreover, they can be used to provide personal tutoring [22], [23] that can assist students with their problems, provide explanations on errors [24], and help students overcome writer's block. Such tutoring systems enable students to have a real-time dialog [25] about their code and the corrections needed with the artificial tutor.

Despite obvious benefits for teaching and learning, potential challenges were also quickly identified [6], [26]. Automatically generated code raises questions on code reusability and licensing, with unclear practices about how much LLM-generated code students can include in their programs while still attributing the solution as one's own. Other challenges are, for example, related to potential social bias of the LLM-produced code [27], sustainability of tools building on LLMs that incur high energy costs for training [28], and security of generated code, as students may not be able to comprehensively evaluate this aspect; nevertheless, code produced by LLMs can actually be more secure than code produced by humans [29]. In addition to challenges with the resulting code, the probably most discussed problem in education is related to academic misconduct. Using LLMs to solve tasks in introductory programming courses can be tempting to many students who struggle with completing the tasks independently. Early versions of the LLMs received mixed results in solving programming tasks [15], [30], but current versions can solve most tasks associated with a typical programming course, if not all of them [31]. While LLMs have typically excelled in basic programming tasks, they seem to be able to outperform typical students in more complex coding problems as well [32]. Researchers have explored ways of detecting LLM-generated code [9] and creating more LLM-resistant assignments [33], but while small changes in the problems or their wording can affect the models' ability to solve the problems [34], the models are improving so quickly that this battle is likely already lost.

Methods for identifying LLM-generated code from human-generated code have been proposed. For example, AI-generated code have less perplexity, which allows the utilization of perturbation to identify them [35]. However, the results seem mixed: it seems that the current, publicly available AI-generated code detectors in fact perform quite poorly [36]. However, machine learning models trained explicitly to identify LLM-generated code can perform quite well, achieving over 90 % accuracy in CS1 context [9]. Code stylometry and complexity [37] are typical examples of metrics used to detect whether code was generated by an LLM or written by a human.

B. Academic honesty in online courses

Maintaining academic honesty is particularly challenging in online settings, such as MOOCs (Massively Open Online Courses) as they always include the risk of students not solving assignments on their own. Still, offering such courses is important as a means of providing flexible and equal opportunities for continuous learning in our fast-evolving society. While LLMs further exacerbate the risk and temptation of cheating by making it easier and more accessible, concerns about cheating and plagiarism in MOOCs were raised already before the rise of LLMs [38]. Moreover, the sudden move to online teaching during the Covid-19 pandemic saw a rise in cheating [39], [40]. After the introduction of LLMs, studies show that students can complete and pass university courses merely based on using such tools [41].

There are methods for trying to prevent or detect cheating. Proctoring systems [42] can, for example, limit access to web services, provide real-time monitoring, utilize facial detection, or analyze students' answers on the fly, e.g. using keystroke dynamics [43]. Naturally, such services are mainly usable during exams (if even then) due to ethical and technical issues. Another, very different solution is to change students' attitudes and knowledge about the use of AI tools [44] by providing clear guidelines and expectations and engaging discussions when possible. Other concrete examples [45] include using complex, open-ended problems, oral examinations, or group projects. However, all these methodologies should be thoroughly studied in order to evaluate their effectiveness. Moreover, many of the proposed solutions are difficult or impossible to implement in MOOC settings with a large number of students working remotely.

III. METHODOLOGY

A. Context

The context of this study is two introductory programming MOOCs, provided for free by the University of Helsinki, located in Finland. Both MOOCs are 5 ECTS credits[1], and have been developed for computer science freshmen. One of the MOOCs uses Java, while the other uses Python. Both of the courses are available in English. Conceptually, the Python version was intended as a replacement for the Java version, but both courses have been retained as the Java version continues to be popular.

B. Data

Students attending the courses can provide research consent, allowing the use of their submissions for research. For this study, we obtained student assignment submissions from students with research consent from the years 2022 and 2024.

To be able to compare whether possible changes are present also across the courses, our analysis focuses mainly on assignments that are in both courses. As these shared tasks are relatively small in scope, we also include a larger course-specific assignments from each course. The assignments are briefly described in Table I.

For each assignment, we sampled submissions from up to 2500 students with research consent. If there were fewer students with research consent for a specific assignment, for the assignment, we included all students with research consent.

[1]European Credit Transfer System; one ECTS credit corresponds to approximately 27 hours of work

TABLE I
BRIEF DESCRIPTIONS OF THE COURSE ASSIGNMENTS INCLUDED IN THE STUDY. THE ASSIGNMENT HANDOUTS TYPICALLY COME WITH INPUT AND OUTPUT EXAMPLES. THE CARDPAYMENT AND BIGYEAR ASSIGNMENTS ARE SPECIFIC TO THE JAVA COURSE, WHILE THE OWNLANGUAGE ASSIGNMENT IS SPECIFIC FOR THE PYTHON COURSE.

Assignment	Description
LeapYear	Write a program that reads a number from the user and prints information on whether the year is a leap year or not.
GiftTax	Write a program that reads a donation amount (a number) and uses a set of rules to determine how much the donation would be taxed.
Numbers	Write a program that reads in numbers from the user until the user types -1. Then, print the sum of the numbers, count of the numbers, the average of the numbers, and the amount of even and odd numbers.
Stars	Write a program that reads in a number from the user and prints a christmas tree that has the height of the number entered by the user.
ListAverage	Write a program that prints the average of the numbers in a given list.
CardPayment	Object-oriented programming assignment where the student implements two classes, one for a money card and one for a terminal that uses the card.
BigYear	End-of-course assignment where the student implements a program for tracking bird observations.
OwnLanguage	End-of-course assignment where the student implements their own programming language.

Altogether, the dataset has submissions from 12,297 students with research consent.

C. Analyses

In our analyses, we treat 2022 data as 'pre-LLM' data, and the 2024 as 'post-LLM' data. We classify 2022 as pre-LLM data since the release of ChatGPT in late 2022 marked a significant shift in student engagement with LLMs, as also indicated in prior work [11].

To answer RQ1, *How has the number of submissions and submission code lengths changed over time in the courses?*, we extract the submission counts and the lines of codes (LOC) for the 2022 and 2024 course iteration assignment submissions. Then, we apply the non-parametric Kruskal-Wallis H test [46] to assess whether there are differences between the assignment-specific statistics between the 2022 and 2024 course iterations.

To answer RQ2, *How has the stylistics of code submissions changed over time?*, we extract the number of commented lines in the source code and the length of the variable names in terms of characters. These metrics are compared across the two datasets using the Kruskal-Wallis H test.

Finally, to answer RQ3, *How has the complexity and originality of exercise submissions changed over time?*, we extract McCabe's cyclomatic complexity [47] for each submission and compare the difference using Kruskal-Wallis H test. Additionally, we examine the differences in code structure using Jensen-Shannon divergence [48]. To do this, we first convert each submitted source code into a set of tri-grams (n-grams with n=3) via text vectorization, then measure the divergence between their probability distributions.

Together, these analyses provide a multi-dimensional evaluation of student programming practices before and after the emergence of large language models. This allows us to assess changes in quantity, quality, and structural complexity of code submissions.

D. Statement on statistically significant differences

Our analyses does not apply a correction for multiple comparisons, such as the Bonferroni correction, because each statistical test addresses a distinct research question with its own outcome measures. Over-correction in this context could unnecessarily inflate the risk of Type II errors, thereby obscuring potentially meaningful differences. Moreover, as noted by the American Statistical Association, researchers are encouraged to move beyond simply interpreting p-values and to instead consider the broader evidential context to create a more nuanced understanding of the results [49]. Effect sizes are reported using epsilon squared (ϵ^2) for Kruskal-Wallis H test; following [50], we interpret values smaller than 0.01 as negligible, and values between 0.01 and 0.04 as weak.

IV. RESULTS

A. Submission counts and code lengths

Table II outlines the average submissions per student, average submissions lengths (in tokens), and average lines of code (LOC) for the assignments over the two years, and the corresponding p-values for the Kruskal-Wallis H test. When considering the submissions per student, across all assignments, the average number of submissions per student are very similar between 2022 and 2024. However, for some Java assignments, there is a statistically significant decrease in submission counts. The effect sizes, in terms of ϵ^2, are negligible, 0.003 for LeapYear, 0.002 for Stars, and 0.001 for CardPayment.

Similar to submissions per student, the submission lengths (in terms of tokens) are also very similar between the two years. There are, however, a handful of assignments where the differences are statistically significant. For example, the average submission lengths for LeapYear are smaller for both programming languages, although the effect size ϵ^2 is again negligible, 0.004 for Python and 0.001 for Java. The effect sizes are negligible also for the Python assignments GiftTax (0.002), ListAverage (0.001), and OwnLanguage (0.004).

There are also minor differences between the lines of code between the years. Similar to the previous results, however, the differences are negligible.

B. Stylistics

Statistics on submission stylistics (commented lines, average variable name length) are shown in Table III. Overall, for the average number of commented lines per submission, there is a slight increase in many assignments, where the differences are statistically significant for three Python assignments and one Java assignment. For these, for Java, there is a slight decrease in the number of comments, while for Python, there is a slight increase in comments. Despite the average number of

TABLE II
AVERAGE SUBMISSIONS PER STUDENT, AVERAGE SUBMISSION LENGTHS (IN TOKENS), AND AVERAGE SUBMISSION LENGTHS IN LINES OF CODE (LOC) FOR THE ASSIGNMENTS BETWEEN THE YEARS 2022 AND 2024.

		Submissions per student			Submission length (tokens)			LOC		
Assignment	Lang.	2022	2024	p_{KW}	2022	2024	p_{KW}	2022	2024	p_{KW}
LeapYear	Python	1.18	1.16	0.534	121.53	120.95	<**0.001**	11.14	10.95	0.068
	Java	2.99	2.53	<**0.001**	53.31	52.42	**0.008**	21.08	20.90	**0.010**
GiftTax	Python	1.15	1.14	0.618	122.31	120.37	**0.006**	21.01	21.03	0.873
	Java	1.91	1.86	0.266	240.76	240.36	0.329	31.06	31.19	**0.024**
Numbers	Python	1.16	1.17	0.360	83.85	83.26	0.536	24.32	24.79	**0.034**
	Java	1.16	1.17	0.360	177.83	177.97	0.714	41.58	41.60	0.400
Stars	Python	1.25	1.22	0.301	76.41	76.26	0.895	16.99	17.13	0.459
	Java	4.20	3.71	**0.002**	231.58	234.52	0.387	51.73	51.64	0.989
ListAverage	Python	1.08	1.07	0.564	36.96	37.40	**0.031**	7.05	7.12	0.192
	Java	1.29	1.26	0.339	140.73	140.63	0.805	31.95	31.98	0.549
CardPayment	Java	3.83	3.53	**0.043**	128.69	129.67	0.700	38.29	38.60	0.543
BigYear	Java	1.70	1.67	0.829	183.83	185.34	0.575	44.68	44.28	0.320
OwnLanguage	Python	2.30	2.34	0.151	631.52	589.76	**0.049**	108.64	105.79	0.396

comments increasing noticeably for some of the assignments (e.g. from 4.05 to 6.14 for Numbers), the effect sizes ϵ^2 are negligible.

Similar observations can be made for the average length of the variable names, where some statistically significant changes exist. The differences are more visible for Python (4 out of 6 assignments) than for Java (1 out of 7 assignments). For Python, overall, for the affected assignments, the average variable name length increases. However, the effect sizes are again negligible.

C. Complexity and originality

Statistics on McCabe's cyclomatic complexity and code originality (measured using Jensen Shannon divergence from tri-grams, JSD_3) are shown in Table IV. Overall, the changes in cyclomatic complexity are minor in absolute terms (between 0.1-0.3 points). A handful of assignments show statistically significant differences, although like in the previous analyses, the differences are negligible.

When considering the code originality, the JSD_3 metric is small for the majority of the Java assignments, which indicates stability in the local syntactic patterns of the code submissions. However, several Python assignments showcase moderate to high JSD_3 values (up to 0.28 for OwnLanguage).

V. DISCUSSION

A. Summarizing answers to research questions

Overall, our results highlight that although there are some statistically significant differences between the years for the course assignments, the differences are – in terms of ϵ^2 – negligible.

For RQ1, *How has the number of submissions and submission code lengths changed over time in the courses?*, for Java, we see a tendency for statistically significant reductions in submission counts in 2024. For Python, we observe significant differences in token counts for a few assignments, although the submission counts remain largely unchanged (and small from the start). For RQ1, in general, with the exception of submission counts for Java, there seems to be a high degree of consistency in submission behavior and high-level code structure across the two years. The differences that are statistically significant are not substantial.

For RQ2, *How has the stylistics of code submissions changed over time?*, for Python, there is a statistically significant increase in the number of commented lines and variable name lengths for many of the assignments. Although the differences are often relatively small, they are to some extent visible – even to the extent that the average number of comments increases by 50% for one of the assignments. For Java, the stylistics are largely unchanged between the years – as an example, for the assignment where the number of comments in Python rose by 50%, there is a less than 1% change in comments in the Java counterpart. More broadly, the differences between Python and Java indicate that the stylistic changes are not necessarily uniform across the two languages, with Python showing more stylistic changes between 2022 and 2024.

For RQ3, *How has the complexity and originality of exercise submissions changed over time?*, the changes in average cyclomatic complexity are largely very small between the years across the programming languages. When considering the JSD_3 metric, the Java submissions seem to be fairly stable between the years, while the Python submissions show more differences. Broadly speaking, the token-level patterns and coding styles for Python seem to have experienced a more notable change between the two years.

B. On small effect sizes

We note that it is possible that the statistically significant differences stem from the large sample size. With thousands of submissions, even small differences in metrics like token count can become statistically detectable. This is, in part, why researchers have suggested moving beyond p-values [49], and including additional contextual evidence like effect sizes.

In our case, all of the effect sizes were negligible, even though at an extreme, there was an increase of 50% in the

TABLE III
AVERAGE NUMBER OF COMMENTED LINES AND AVERAGE LENGTH OF VARIABLE TOKENS IN THE ASSIGNMENT SUBMISSIONS BETWEEN THE YEARS 2022 AND 2024.

		Commented lines			Variable name length		
Assgn.	Lang.	2022	2024	p	2022	2024	p
LeapYear	Python	10.65	10.97	0.900	4.33	4.33	0.154
	Java	0.66	0.59	0.941	5.27	5.27	0.660
GiftTax	Python	5.16	5.60	0.077	4.66	4.85	**0.004**
	Java	0.39	0.46	0.843	5.27	5.30	0.531
Numbers	Python	4.05	6.14	**0.002**	5.26	5.40	**0.002**
	Java	10.06	9.98	0.170	5.89	5.92	0.454
Stars	Python	16.87	18.45	**0.030**	4.42	4.56	**0.002**
	Java	8.58	8.65	0.247	5.87	5.82	**0.011**
ListAverage	Python	37.93	37.30	0.153	3.16	3.19	0.164
	Java	9.70	9.81	0.871	5.50	5.51	0.134
CardPayment	Java	9.38	9.47	0.173	7.72	7.69	0.804
BigYear	Java	10.96	10.24	**0.003**	6.54	6.55	0.614
OwnLanguage	Python	12.17	13.50	**0.030**	6.13	6.34	**0.009**

TABLE IV
AVERAGE CYCLOMATIC COMPLEXITY AND JENSEN-SHANNNON DIVERGENCE FOR TRIGRAMS (JSD_3) IN THE ASSIGNMENT SUBMISSIONS BETWEEN THE YEARS 2022 AND 2024.

		Cyclomatic Complexity			
Assgn.	Lang.	2022	2024	p	JSD_3
LeapYear	Python	4.84	4.74	<**0.001**	0.036
	Java	4.58	4.53	**0.005**	0.012
GiftTax	Python	8.80	8.52	<**0.001**	0.109
	Java	9.00	8.97	0.133	0.041
Numbers	Python	4.63	4.68	**0.031**	0.117
	Java	4.09	4.07	0.946	0.025
Stars	Python	3.28	3.26	0.326	0.166
	Java	9.36	9.47	<**0.001**	0.025
ListAverage	Python	1.98	1.99	0.828	0.034
	Java	3.93	3.92	0.554	0.022
CardPayment	Java	5.52	5.56	0.499	0.013
BigYear	Java	7.51	7.52	0.718	0.054
OwnLanguage	Python	24.93	25.16	0.372	0.283

average number of comments in one of the assignments. There are a handful of possible explanations for why the effect sizes might be small.

First, in a MOOCs, the participating population is typically diverse in terms of student backgrounds, skills, and coding styles. This heterogeneity of the course populations can contribute to a high within-group variability. This, in turn, can result in a high overlap between the groups, and consequently dilute the between-group effects, leading to small effect sizes despite significant differences.

Second, the assignments are highly structured and come with strict guidelines that allow automated assessment of the submitted solutions. The assignment structure and guidelines may limit the variation in student submissions, which can further reduce the differences between the groups. In a way, we might be seeing a "ceiling effect", where the allowable solutions are so constrained where even using an LLM in the process may only produce minor changes in the observable code – such as changes in number of comments.

Third, as the participating population is typically diverse, it is also possible that the adoption of LLM-based coding assistants also differs between the student populations. The 2024 dataset – and perhaps parts of the 2022 dataset – likely includes a mix of LLM-assisted and traditional code submissions. This mixture can again dilute the observed differences, where the observed differences are minor across the entire population.

Fourth, as LLMs have been trained with data that is publicly available, including questions on platforms like StackOverflow and Reddit, LLM-based coding assistants have also learned patterns that are present in student code submissions. The assignments of the MOOCs have also received plenty of questions on the said platforms. This can further lead to a situation where the differences between the years are not major, in part as the models may already have knowledge of sample solutions. This problem is further exacerbated by the constraints set by the assignments.

Taking all of the above into account, the observed negligible effect sizes might thus be in part due to the specific nature of our data, and greater between-groups effects might be observed with data from other (e.g. closed) courses.

C. Detecting LLM-generated submissions

When considering prior studies that have compared LLM-generated submissions and student submissions, the studies have mainly focused on differences between human-submitted codes and LLM-generated codes [9], [37] where it has been known which codes are human and which LLM-generated. In our case, we compare overall data from the courses, where differences may be harder to detect as it is likely that only a minority of the student submissions are (entirely) LLM-generated. This could explain why our effect sizes are low – any differences are harder to tease out when most of the data is similar (i.e., human-generated). In future work, it would be interesting to analyze the submissions for patterns that could indicate LLM-generated code, such as using ternary operators and code that is "too optimized" considering typical code produced by novice students [9]. At the same time, such endeavors must be planned carefully so as to ensure any observed effects are not false positives resulting from how the data is split into 'normal' and 'non-normal' groups.

D. Limitations of the study

This study comes with a range of limitations, which we acknowledge here. First and foremost, we do not have a ground truth on whether students in the MOOCs have used LLMs when working on their course submissions. The courses have not, however, been adjusted between 2022 and 2024, which means that any observed changes between the years should not be attributed to possible course changes. However, the student populations can be different between the years, which could partially explain our findings.

Second, the data for the study comes from a single institution that offers two programming MOOCs. Institution-specific practices, including course design and especially the amount of scaffolding, may influence submission behaviors. Thus, the results might differ in courses from other institutions or educational settings.

Third, the study included only students who provided research consent and who submitted solutions to the assignments in a MOOC. This suggests that the data suffers also from self-selection bias, which means that the participant sample might not be entirely representative of the broader student population (and possibly, even the student population in the MOOCs). As an example, it is possible that students who overuse LLM support could be less likely to provide research consent.

Fourth, our analysis focuses on quantitative code metrics such as lines of code, token counts, cyclomatic complexity, and stylistic measures. The metrics capture only surface-level differences and do not, for example, provide evidence of (lack of) deeper understanding.

Fifth, we acknowledge that the large sample size can produce statistically significant p-values, where the differences can be in reality negligible. We have discussed some of the potential reasons for the negligible results in Section V-B.

Finally, as also acknowledged in Section V-B, it is possible that the structured nature of the MOOCs can also influence the observations. The focus on introductory programming also constraints the complexity of the assignments. For nearly all assignments, the average cyclomatic complexity was under 10 – coincidentally, this is also in line with what McCabe calls a reasonable upper limit for code complexity after which the code should be modularized [47].

VI. Conclusion

In this work, we examined the differences in student submissions before (2022) and after (2024) the release of ChatGPT in two different large, introductory programming courses, one in Python and one in Java. Our findings suggest that – against our expectations – the differences between the years are minor. While there are multiple statistically significant results, such as specific exercises having more comments and longer variable names since the release of ChatGPT, the effect sizes are very low. Potential reasons for these findings could be, for example, that only a minority of students use LLMs for solving exercises, that the constrained nature of the exercises hides any potential LLM use, or that LLM-generated code is more similar to student code than we thought.

Acknowledgements

This work was in part supported by Research Council of Finland grants #356114 and #367787. We would like to acknowledge the help of Antti Leinonen in preparing the data for this study.

References

[1] D. R. E. Cotton, P. A. Cotton, and J. R. S. and, "Chatting and cheating: Ensuring academic integrity in the era of ChatGPT," *Innovations in Education and Teaching International*, vol. 61, no. 2, pp. 228–239, 2024.

[2] P. Denny, J. Leinonen, J. Prather, A. Luxton-Reilly, T. Amarouche, B. A. Becker, and B. N. Reeves, "Prompt problems: A new programming exercise for the generative AI era," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pp. 296–302, 2024.

[3] B. N. Reeves, J. Prather, P. Denny, J. Leinonen, S. MacNeil, B. A. Becker, and A. Luxton-Reilly, "Prompts first, finally," *arXiv preprint arXiv:2407.09231*, 2024.

[4] S. Lau and P. Guo, "From "ban it till we understand it" to "resistance is futile": How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot," in *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*, pp. 106–121, 2023.

[5] Y. Fan, L. Tang, H. Le, K. Shen, S. Tan, Y. Zhao, Y. Shen, X. Li, and D. Gašević, "Beware of metacognitive laziness: Effects of generative artificial intelligence on learning motivation, processes, and performance," *British Journal of Educational Technology*, vol. 56, no. 2, pp. 489–530, 2025.

[6] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "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*, pp. 500–506, 2023.

[7] J. Prather, B. N. Reeves, J. Leinonen, S. MacNeil, A. S. Randrianasolo, B. A. Becker, B. Kimmel, J. Wright, and B. Briggs, "The widening gap: The benefits and harms of generative ai for novice programmers," in *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*, pp. 469–486, 2024.

[8] J. L. Weber, B. M. Neda, K. C. Juarez, J. Wong–Ma, S. Gago–Masague, and H. Ziv, "Beyond the hype: Perceptions and realities of using large language models in computer science education at an R1 university," in *2024 IEEE Global Engineering Education Conference (EDUCON)*, pp. 01–08, 2024.

[9] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch, T. Price, and B. Akram, "Detecting ChatGPT-generated code submissions in a cs1 course using machine learning models," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pp. 526–532, 2024.

[10] K. Z. Zhou, Z. Kilhoffer, M. R. Sanfilippo, T. Underwood, E. Gumusel, M. Wei, A. Choudhry, and J. Xiong, ""the teachers are confused as well": A multiple-stakeholder ethics discussion on large language models in computing education," 2024.

[11] J. Prather, P. Denny, J. Leinonen, B. A. Becker, I. Albluwi, M. Craig, H. Keuning, N. Kiesler, T. Kohn, A. Luxton-Reilly, *et al.*, "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*, pp. 108–159, 2023.

[12] J. Prather, J. Leinonen, N. Kiesler, J. Gorson Benario, S. Lau, S. MacNeil, N. Norouzi, S. Opel, V. Pettit, L. Porter, *et al.*, "Beyond the hype: A comprehensive review of current trends in generative AI research, teaching practices, and tools," *2024 Working Group Reports on Innovation and Technology in Computer Science Education*, pp. 300–338, 2025.

[13] C. Zastudil, M. Rogalska, C. Kapp, J. Vaughn, and S. MacNeil, "Generative AI in computing education: Perspectives of students and instructors," in *2023 IEEE Frontiers in Education Conference (FIE)*, pp. 1–9, IEEE, 2023.

[14] L. Leppänen, L. Aunimo, A. Hellas, J. K. Nurminen, and L. Mannila, "Emergence of LLMs: (not-so-)significant delving in essay answers in a MOOC on the ethics of AI," in *Artificial Intelligence in Education* (A. I. Cristea, E. Walker, Y. Lu, O. C. Santos, and S. Isotani, eds.), (Cham), pp. 36–43, Springer Nature Switzerland, 2025.

[15] P. Denny, J. Prather, B. A. Becker, J. Finnie-Ansley, A. Hellas, J. Leinonen, A. Luxton-Reilly, B. N. Reeves, E. A. Santos, and S. Sarsa, "Computing education in the era of generative AI," *Communications of the ACM*, vol. 67, no. 2, pp. 56–67, 2024.

[16] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, and A. Hellas, "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*, pp. 124–130, 2023.

[17] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, and J. Leinonen, "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*, pp. 931–937, 2023.

[18] B. Jury, A. Lorusso, J. Leinonen, P. Denny, and A. Luxton-Reilly, "Evaluating LLM-generated worked examples in an introductory programming course," in *Proceedings of the 26th Australasian computing education conference*, pp. 77–86, 2024.

[19] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "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*, pp. 27–43, 2022.

[20] E. Logacheva, A. Hellas, J. Prather, S. Sarsa, and J. Leinonen, "Evaluating contextually personalized programming exercises created with generative AI," in *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*, pp. 95–113, 2024.

[21] A. Del Carpio Gutierrez, P. Denny, and A. Luxton-Reilly, "Evaluating automatically generated contextualised programming exercises," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pp. 289–295, 2024.

[22] K. Chrysafiadi, M. Virvou, G. A. Tsihrintzis, and I. Hatzilygeroudis, "Evaluating the user's experience, adaptivity and learning outcomes of a fuzzy-based intelligent tutoring system for computer programming for academic students in Greece," *Education and Information Technologies*, vol. 28, no. 6, pp. 6453–6483, 2023.

[23] R. Thinakaran and S. Chuprat, "Students' characteristics of student model in intelligent programming tutor for learning programming: a systematic literature review," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 7, 2022.

[24] J. Leinonen, A. Hellas, S. Sarsa, B. Reeves, P. Denny, J. Prather, and B. A. Becker, "Using large language models to enhance programming error messages," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pp. 563–569, 2023.

[25] J. Perez and O. Ethel, "Designing an LLM-based dialogue tutoring system for novice programming," in *International Conference on Computers in Education*, 2024.

[26] P. Denny, B. A. Becker, J. Leinonen, and J. Prather, "Chat overflow: Artificially intelligent models for computing education-renAIssance or apocAIypse?," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, pp. 3–4, 2023.

[27] L. Ling, F. Rabbi, S. Wang, and J. Yang, "Bias unveiled: Investigating social bias in LLM-generated code," *arXiv preprint arXiv:2411.10351*, 2024.

[28] A. Faiz, S. Kaneda, R. Wang, R. Osi, P. Sharma, F. Chen, and L. Jiang, "Llmcarbon: Modeling the end-to-end carbon footprint of large language models," *arXiv preprint arXiv:2309.14393*, 2023.

[29] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[30] S. I. Salim, R. Y. Yang, A. Cooper, S. Ray, S. Debray, and S. Rahaman, "Impeding LLM-assisted cheating in introductory programming assignments via adversarial perturbation," *arXiv preprint arXiv:2410.09318*, 2024.

[31] J. Rytilahti and E. Kaila, "How easy is it to cheat?-solving programming exercises automatically with AI," *Appeared in CDIO 2024 Conference*, 2024.

[32] J. Finnie-Ansley, P. Denny, A. Luxton-Reilly, E. A. Santos, J. Prather, and B. A. Becker, "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*, pp. 97–104, 2023.

[33] D. M. Popescu and D. A. Joyner, "ChatGPT's performance on problem sets in an at-scale introductory computer science course," in *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, L@S '24, (New York, NY, USA), p. 486–490, Association for Computing Machinery, 2024.

[34] T. T. Dang, H.-C. Ling, and N. Q. Tran, "Combating ChatGPT-based programming test cheating—an evaluation using public problems," in *2024 6th International Conference on Computer Science and Technologies in Education (CSTE)*, pp. 161–165, IEEE, 2024.

[35] Z. Xu and V. S. Sheng, "Detecting AI-generated code assignments using perplexity of large language models," in *Proceedings of the aaai conference on artificial intelligence*, vol. 38, pp. 23155–23162, 2024.

[36] W. H. Pan, M. J. Chok, J. L. S. Wong, Y. X. Shin, Y. S. Poon, Z. Yang, C. Y. Chong, D. Lo, and M. K. Lim, "Assessing AI detectors in identifying AI-generated code: Implications for education," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, pp. 1–11, 2024.

[37] O. J. Idialu, N. S. Mathews, R. Maipradit, J. M. Atlee, and M. Nagappan, "Whodunit: Classifying code as human authored or GPT-4 generated-a case study on CodeChef problems," in *Proceedings of the 21st International Conference on Mining Software Repositories*, pp. 394–406, 2024.

[38] E. Surahman and T.-H. Wang, "Academic dishonesty and trustworthy assessment in online learning: A systematic literature review," *Journal of Computer Assisted Learning*, vol. 38, no. 6, pp. 1535–1553, 2022.

[39] S. Janke, S. C. Rudert, Änne Petersen, T. M. Fritz, and M. Daumiller, "Cheating in the wake of COVID-19: How dangerous is ad-hoc online testing for academic integrity?," *Computers and Education Open*, vol. 2, p. 100055, 2021.

[40] B. D. Jenkins, J. M. Golding, A. M. L. Grand, M. M. Levi, and A. M. Pals, "When opportunity knocks: College students' cheating amid the COVID-19 pandemic," *Teaching of Psychology*, vol. 50, no. 4, pp. 407–419, 2023.

[41] J. Bock, R. D. Schmallegger, M. McKay, M. Meadows, and S. Holzer, "Using ChatGPT to pass online courses: Lessons learned in higher ed," *Advances in Online Education: A Peer-Reviewed Journal*, vol. 3, no. 1, pp. 69–84, 2024.

[42] T. L. Dang, N. M. N. Hoang, T. V. Nguyen, H. V. Nguyen, Q. M. Dang, Q. H. Tran, and H. H. Pham, "Auto-proctoring using computer vision in MOOCs system," *Multimedia Tools and Applications*, pp. 1–27, 2024.

[43] J. Leinonen, K. Longi, A. Klami, A. Ahadi, and A. Vihavainen, "Typing patterns and authentication in practical programming exams," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 160–165, 2016.

[44] J. Cantiello and R. H. Geschke, "Preventing academic dishonesty in online courses: Best practices to discourage cheating," *Journal of Health Administration Education*, vol. 40, no. 2, pp. 205–230, 2024.

[45] Y. Xie, S. Wu, and S. Chakravarty, "AI meets AI: Artificial intelligence and academic integrity-a survey on mitigating AI-assisted cheating in computing education," in *Proceedings of the 24th annual conference on information technology education*, pp. 79–83, 2023.

[46] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[47] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[48] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 1991.

[49] R. L. Wasserstein and N. A. Lazar, "The asa statement on p-values: context, process, and purpose," 2016.

[50] L. M. Rea and R. A. Parker, *Designing and conducting survey research: A comprehensive guide*. John Wiley & Sons, 2014.