The Robots are Here: Navigating the Generative AI Revolution in Computing Education

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

Paul Denny*
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

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

Brett A. Becker*
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Ibrahim Albluwi
Princess Sumaya University for Technology
Amman, Jordan
i.albluwi@psut.edu.jo

Michelle Craig
University of Toronto
Toronto, Canada
mcraig@cs.toronto.edu

Hieke Keuning
Utrecht University
Utrecht, The Netherlands
h.w.keuning@uu.nl

Natalie Kiesler
DIPF Leibniz Institute for Research and Information in Education
Frankfurt am Main, Germany
kiesler@dipf.de

Tobias Kohn
Karlsruhe Institute of Technology
Karlsruhe, Germany
tobias.kohn@kit.edu

Andrew Luxton-Reilly
University of Auckland
Auckland, New Zealand
andrew@cs.auckland.ac.nz

Stephen MacNeil
Temple University
Philadelphia, Pennsylvania, USA
stephen.macneil@temple.edu

Andrew Petersen
University of Toronto Mississauga
Mississauga, Canada
andrew.petersen@utoronto.ca

Raymond Pettit
University of Virginia
Charlottesville, Virginia, USA
raymond.pettit@virginia.edu

Brent N. Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu

Jaromir Savelka
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
jsavelka@cs.cmu.edu

ABSTRACT

Recent advancements in artificial intelligence (AI) and specifically generative AI (GenAI) are threatening to fundamentally reshape computing and society. Largely driven by large language models (LLMs), many tools are now able to interpret and generate both natural language instructions and source code. These capabilities have sparked urgent questions in the computing education community around how educators should adapt their pedagogy to address the challenges and to leverage the opportunities presented by this new technology. In this working group report, we undertake a comprehensive exploration of generative AI in the context of computing education and make five significant contributions. First, we provide a detailed review of the literature on LLMs in computing education and synthesise findings from 71 primary articles, nearly 80% of which have been published in the first 8 months of 2023. Second, we report the findings of a survey of computing students and instructors from across 20 countries, capturing prevailing attitudes towards GenAI/LLMs and their use in computing education contexts. Third, to understand how pedagogy is already changing, we offer insights collected from in-depth interviews with 22 computing educators from five continents. Fourth, we use the ACM Code of Ethics to frame a discussion of ethical issues raised by the use of large language models in computing education, and we provide concrete advice for policy makers, educators, and students. Finally, we benchmark the performance of several current GenAI models/tools on various computing education datasets, and highlight the extent to which the capabilities of current models are rapidly improving.

There is little doubt that LLMs and other forms of GenAI will have a profound impact on computing education over the coming years. However, just as the technology will continue to improve, so will our collective knowledge about how to leverage these new models and tools in educational settings. We expect many important conversations around this topic will emerge as the community explores how to provide more effective, inclusive, and personalised learning experiences. Our aim is that this report will serve as a focal point for both researchers and practitioners who are exploring, adapting, using, and evaluating GenAI and LLM-based tools in computing classrooms.

*Randomly-ordered Working Group Co-leaders

ITiCSE-WGR 2023, July 7–12, 2023, Turku, Finland

ACM ISBN 979-8-4007-0405-5/23/07
https://doi.org/10.1145/3623762.3633499

CCS CONCEPTS

• **Computing methodologies → Artificial intelligence**; • **Social and professional topics → Computing education**.

KEYWORDS

AI; artificial intelligence; code generation; ChatGPT; Codex; computer programming; curriculum; Copilot; CS1; Generative AI; GitHub; GPT; GPT-3; GPT-4; large language models; LLM; LLMs; novice programming; OpenAI; pedagogical practices; programming

ACM Reference Format:
James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2023), July 7–12, 2023, Turku, Finland*. ACM, New York, NY, USA, 52 pages. https://doi.org/10.1145/3623762.3633499

1 INTRODUCTION

Many disruptions to computing education—and education globally—have occurred in the past few years. During the COVID-19 pandemic, students adapted to learning online in unprecedented ways. It was during this time Generative AI became available to the public with the November 2022 release of ChatGPT being the main catalyst. Suddenly, students are not just learning *about* AI in advanced computer science courses, but *using* it—possibly in many courses, including introductory ones. Unlike before, they are not only using it passively where AI powers some aspect of the tools they might use (such as Google Translate where AI *transforms* data), but in an active manner where students are knowingly and intentionally using and interacting with AI as a tool to *generate* new data with natural language prompts. These generative tools have much broader capabilities than what was available just a few years ago and can be used in all disciplines including computing for myriad tasks.

In computing education, researchers have demonstrated that these models have an increasing capacity to perform source code generation and interpretation through a natural language interface [74]. For instance, it is likely that pair programming might evolve in some cases from two students working together to a student and their LLM working together [68]. Many of these models are easily available and free for students, and early reports reveal that students are already using them for assistance on their assignments [44]. In addition, there is now at least one textbook, published in September 2023, which features the use of Generative AI—specifically GitHub Copilot and ChatGPT—from day 1 of introductory programming courses [155]. The profound impacts of LLMs on computing education are still not entirely known but are already being felt by educators [121].

The evidence gathered over the past few decades about how students learn best supports the commonly adopted approach of having students write many small programs checked by automated assessment tools over the course of their introductory terms [23]. While other methods of learning programming exist, such as constructivist approaches [40], the "many small programs" approach has remained nearly ubiquitous for decades. However, this approach may have become obsolete given how easily most LLMs can now solve introductory computing problems with simple prompts [85, 86, 167, 175]. Furthermore, generative AI models can provide wrong or biased answers, and students may also become over-reliant on LLM tools or generate code plagiarised from online sources by the model [33]. The models might generate code students do not understand [108] or may distract them with large blocks of text they did not write [159]. Teachers may look to AI detectors to enforce some semblance of normal, but evidence is mounting that these tools are currently ineffective [144]. However, these models offer computing educators opportunities in addition to the aforementioned challenges. Recent research has shown promising possibilities for providing students with LLM partners in pair programming, given the right context and with the right scaffolding and support [44, 108, 159]. LLMs can also provide detailed code explanations to support students working through difficult problems [123, 137] and can even explain error messages [125] known to have vexed students for decades [36]. Instructors can also benefit as these models can rapidly generate new and personalised teaching materials and programming assignments [75, 174]. Most exciting are the opportunities for entirely new types of programming problems utilising LLMs, such as Prompt Problems [71].

Large language models will have a profound impact on computing education in the next decade as the technology matures and as teachers and researchers identify opportunities. LLMs will change how, what, and whom we teach not only in computing but in all of education [68]. This working group[1] report aims [157] to summarise these early movements in computing education to set an agenda for researchers and to collect effective practices for educators.

1.1 Contributions

This working group report describes the following efforts that, taken together, aim to describe the current state of generative AI in computing education from several angles, and to set out a vision of the future of programming education in the generative AI era:

(1) **Reviewing Literature (Section 2):** We review the existing literature on LLMs in computing education[2] and present a guide to the opportunities and challenges of LLMs in this domain.
(2) **Evaluating Current Attitudes (Section 3):** We conducted an international survey of students and instructors to obtain their perspectives of LLMs. From this data, we provide a snapshot of current attitudes toward LLMs and their uses.
(3) **Identifying New Instructional Approaches (Section 4):** We interviewed instructors in terms of teaching about and/or using LLMs in the classroom. They provide insight into advantages and disadvantages of using LLMs in computing education.
(4) **Exploring Ethical Implications (Sections 5 & 6):** We perform an evidence-based ethical analysis on the use of LLMs in computing education by evaluating the AI policies of several leading universities in the context of the ACM Code of Ethics. These examples suggest how universities are responding—and may in the future further respond—to the ethical challenges

[1] iticse23-generative-ai.github.io
[2] Through August 2023.

presented by such systems. From this, we furthermore discuss academic integrity issues with LLMs and provide resources for both faculty and students to understand when it may or may not be permissible to utilise LLMs (see Appendix D).

(5) **Encouraging Replication (Section 7):** We replicate prior work using new LLMs, highlighting challenges driven by the speed at which LLMs are improving and with current standards for describing research methods. To encourage comparisons between published work, we identify appropriate, openly available datasets and identify concerns with the quality and type of datasets available.

2 REVIEW OF LITERATURE

The working group aimed to identify prior work that explores how large language models might impact computing education. We recognise that any such attempt in this nascent and rapidly expanding area of research will quickly become out of date but aim to establish the *status quaestionis* of this new research field and to provide recommendations based on the current scholarly discourse. Furthermore, we used the work we found to inform the other activities of the working group listed in Section 1.1.

2.1 Method

We chose to perform a scoping review to rapidly identify gaps and major themes in the literature discussing how large language models can support computing education. We explicitly considered but decided not to perform a systematic review, as the research in this area is evolving quickly and relies heavily on dissemination through non-traditional publication channels such as arXiv. We chose to perform one step of forward and backward snowballing [194] from a set of reference papers that were identified as being currently significant work in the area of large language models in computing education. We decided only one step in the snowballing phase was necessary given the recent advent of large language models in computing education. We conducted two separate phases of forward snowballing, one in May 2023 and one in August 2023, with the aim of including as much of recent work as possible.

2.1.1 Reference papers. We collected a set of reference papers using keyword searches over three databases: (1) ACM Digital Library (Full-Text Collection), (2) Taylor & Francis Online, and (3) IEEE Xplore. These choices were guided by the book "Past, Present and Future of Computing Education Research: A Global Perspective" [26] which includes a chapter on venues that have shaped computing education research (pp 121-150). This chapter lists 13 conference and magazine venues and two journals dedicated to computing education research literature, and our database searches were scoped to cover these venues: ACM SIGCSE Sponsored (SIGCSE Technical Symposium, ITiCSE, ICER, CompEd); ACM SIGCSE In-Cooperation (ACE, Koli Calling, COMPUTE, WiPSCE, CCSC); ACM Journal (TOCE); Taylor and Francis (CSE); and IEEE (FiE, ToE, TLT).

The keywords used included "large language models" and "generative AI" as well as three common models (See list below). Queries were refined as appropriate for the different databases, and filters were used as appropriate when scoping the search, such as filtering by "SIGCSE sponsored" venues in the ACM Digital Library. In addition, the searches were conducted using a filter for dates beginning in January 2021. This start date was chosen based on the technological timeline of LLMs and their relevance to computing education. By January 2021, LLMs, especially with the advent of models like GPT-3 in mid-2020, started gaining significant traction and recognition in broader research and application areas. Furthermore, the integration of such advanced LLMs into computing education, pedagogically and practically, was still in nascent stages. By setting January 2021 the start date of the literature search, we aimed to capture the most recent and relevant research insights right from the outset of substantial scholarly attention towards the intersection of LLMs and computing education. As an example, the final query used when searching the ACM Digital Library was:

```
[All: "large language models"] OR
[All: "generative AI"] OR
[All: "Codex"] OR
[All: "GPT-3"] OR
[All: "GPT-4"]
```

The search was conducted on 26th April 2023 and resulted in 19 papers. For each paper, the following inclusion criteria was applied:

(1) *Must mention generative AI, large language models, or a specific tool using those technologies, such as GitHub Copilot.*
(2) *At least 4 pages in length (inclusive).*[3]
(3) *Written in English.*

A total of 3 papers were excluded based on length and 5 for not being aligned with the topic. The resulting set of reference papers ("seed papers"), listed in Table 1, includes 10 papers. By necessity due to the age of the research area, these papers are largely published in 2022 and 2023.

2.1.2 Snowballing (phase 1). Each paper that cites or is cited by at least one of the reference papers was evaluated for inclusion by two working group members. The backward snowballing phase, which considered all papers in the reference list for each paper in the reference set, resulted in 381 papers. For forward snowballing, we used the "cited by" feature in Google Scholar at the beginning of May 2023, resulting in 132 papers. There were duplicates in this list, but we decided not to identify duplicates until the final review. Each of these 513 snowballed papers were assigned to two members of the group. At this stage of the review, the papers were not read in detail; rather, the evaluators searched for evidence that a paper should be given deeper consideration.

2.1.3 Snowballing (phase 2). We ran a second phase of forward snowballing using the "cited by" feature in Google Scholar at the end of August 2023. In the second forward snowballing, a total of 353 new papers (including duplicates) had cited the seed papers. Across all three snowballing phases (two forward, one backward), a total of 866 papers were identified for filtering, i.e. removing duplicates and applying inclusion and exclusion criteria.

2.1.4 Inclusion and Exclusion Criteria. The inclusion criteria included the three criteria used to filter the reference papers plus a publication date criterion and a content criterion:

(1) *Must mention generative AI, large language models, or a specific tool using those technologies, such as Copilot, AND*

[3]This criterion rules out posters and abstracts.

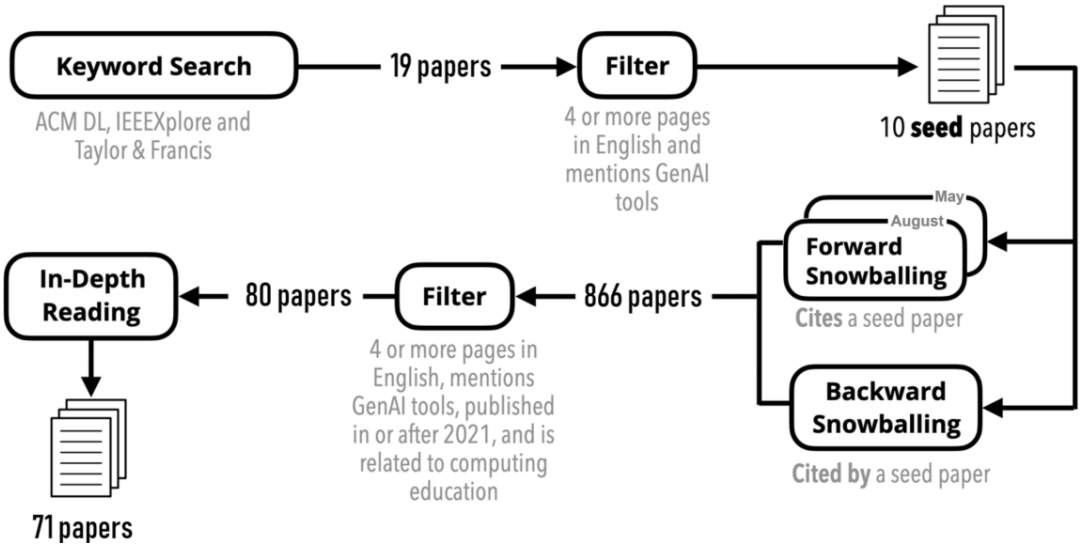


Figure 1: Phases of the literature review.

Table 1: Reference papers used to seed the literature review.

Title	Venue	Year	Citation
The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming	ACE	2022	[85]
Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models	ICER	2022	[174]
Github copilot in the classroom: learning to code with AI assistance	JCSC	2022	[160]
Programming Pedagogy and Assessment in the Era of AI/ML: A Position Paper	COMPUTE	2022	[165]
My AI Wants to Know If This Will Be on the Exam	ACE	2023	[86]
Using Large Language Models to Enhance Programming Error Messages	SIGCSE TS	2023	[125]
Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book	SIGCSE TS	2023	[137]
Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language	SIGCSE TS	2023	[70]
Using GitHub Copilot to Solve Simple Programming Problems	SIGCSE TS	2023	[191]
Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation	SIGCSE TS	2023	[33]

(2) *At least 4 pages in length (inclusive), AND*
(3) *Written in English, AND*
(4) *"Published" in or after 2021.* For papers published in non-traditional venues such as arXiv, this is the upload date, AND
(5) *Must have direct applicability to computing education.* This criterion was refined to the following and was interpreted generously:
   (a) *The paper explicitly states a relation to computing education, OR*
   (b) *the participants include students working on problems typical of a computing education context, OR*
   (c) *the problems or inputs featured are drawn from a computing education context, OR*
   (d) *the resource or tool being created is specifically designed for computing education.*

Each of the five criteria had to be satisfied to include the paper. Each paper was independently evaluated twice; the evaluator could flag the paper for inclusion, exclusion, or discussion. If there was disagreement between the two evaluators or if they flagged the paper for discussion, it was evaluated by a third evaluator who made a final decision. In addition, given the subjectivity of criterion (5), all papers that were marked as not being included because of this criterion were reviewed by a third evaluator. As a result, papers were included if (a) both initial evaluators flagged it for inclusion

or (b) if the third evaluator intervened due to a disagreement between the initial reviewers or when they reviewed criterion (5). All evaluators were instructed to interpret the criteria generously, to avoid exclusion of potentially relevant work.

After the filtering was complete, a total of 80 papers were identified for careful reading.

2.1.5 In-Depth Reading. In the final phase, the 80 papers selected for inclusion were assigned to members of the working group for reading (each paper was read by one member). During this in-depth reading phase, 9 papers were flagged as not being relevant. The exclusion of some papers at this stage was expected, as the reviewers had been instructed to identify any *potentially* relevant work. These exclusions left us with 71 relevant papers (10 reference papers, 28 papers from the first snowballing phase, and 33 papers from the second).

The goal of this step was to identify potential impact on future research in the area or on computing education practice. In addition, we extracted some details about the work being performed, such as the location of authors, the type of work published, and evidence of research quality. Data extraction was guided by a set of questions implemented as an online form to help standardise the process. The questions on the form are presented in Appendix A.

The final list of papers (including the 10 original reference papers and papers from the two snowballing phases) is shown in Table 2. Interestingly, while the second snowballing phase covered only a few months, it resulted in a number of papers that is very close to the number of papers that resulted from the first snowballing phase, which covered a period of around two and a half years. Including a second snowballing phase was motivated by the very fast pace at which the literature is growing in this area, which this finding supports.

2.2 Descriptive statistics

Statistics about the papers included in our analysis are presented in Table 3 and Table 4. The work has been presented in a range of venues, including traditional conferences and journals. However, due to rapid changes in this field, a large number of papers were published only on arXiv. Some of this work was later published in a conference or journal, but some only remains visible—and is cited from—that site.

Despite the recency of this area of research, the papers we reviewed also used a wide range of LLMs, which is described in Table 4. The rapid pace of the field is a potential threat, however, to the results being published. For example, the most commonly used LLM considering papers from the first snowballing phase only (i.e. up to May 2023) was Codex, which is now no longer available, and the most recent version of GPT (GPT-4) had only a single piece of research using it. Table 4 shows the results considering all the papers we analysed (i.e. up to August 2023), where the most commonly used LLM has become GPT-3/3.5, and the most recent version of GPT has 11 papers using it.

Table 4 also describes the languages being investigated. The majority of the research focuses on Python, with some work involving Java and C. The table omits languages only explored by one paper in our set; most of these come from a single paper that investigates multiple languages. The fact that Python emerges as the most popular language in this work is not too surprising, however, as popular LLMs such as Codex have been reported to be most proficient in Python [56].

Table 5 contains our evaluation of four quality metrics reported in Hellas et al. [92]. They reported these metrics as part of a review of performance prediction research, so several of their questions are focused on work from that domain. For example, they ask, "is the value being predicted clearly defined?" We selected the most generally applicable questions, and we updated their question about threats to validity to focus specifically on whether they were discussed in an explicit subsection. Compared to their results, we find that the work in this area is reported more clearly in all four aspects measured. In particular, threats to validity are explicitly discussed in the majority, rather than minority, of cases, and slightly more of the work we examined present explicit research questions.

2.3 Classification of literature

The papers we reviewed broadly fall into five categories, with respect to the role that the LLM plays in the study: (i) assessing the performance, capabilities, and limitations of LLMs, (ii) using LLMs to generate teaching materials, (iii) using LLMs to analyse student work (e.g. identifying errors and repairing bugs), (iv) studying the interactions between programmers and LLMs, and (v) position papers and surveys/interviews. Category (i) is by far the largest group, indicating a strong desire to assess the current capabilities and limitations of LLMs in computing education contexts. We acknowledge that some papers would fit into more than one category; in these cases, we classified the paper into the most fitting category.

We now briefly summarise the main contributions of the papers included in our review, organised into these five categories.

2.3.1 Assessing the performance, capabilities and limitations of LLMs (35): More than thirty papers looked into assessing the performance or capabilities of large language models. Most of these looked into the performance of LLMs in generating code, often for programming exercises [28, 48, 57, 61, 70, 85, 86, 126, 152, 153, 160, 161, 167, 177, 184, 191]. Some looked into other types of exercises such as multiple-choice questions [175–177, 189], textbook questions [103], exam questions [79], computational thinking tasks [39], and textual reports [48]. In general, LLMs seem to perform at a level that is equivalent to or better than that of average students, at least for code generation tasks. For other tasks, such as answering MCQs [176], Parsons problems [167] and computational thinking tasks [39], the performance of LLMs is currently not as great. When used to generate questions or hints to support students in solving problems, the research is inconclusive [21, 30, 82, 151, 192], although LLMs are able to provide better explanations of code than students [123], and are able to explain their answers to textbook questions in about half of the cases [103]. A study found that code generated by ChatGPT had enough differences from student code that it could be detected very accurately (between 96-98% accuracy) [99]. However, tools that assess whether a given text was generated by an LLM show a large number of false positives and should not be trusted blindly [144]. LLMs can also be prompted to act as a programming assistant, even ones originally trained solely for code generation [169], but struggle with questions that require

Table 2: Papers included in the literature review (listed alphabetically by author, grouped by publication year). An arrow (→) followed by a citation indicates that an arXiv paper was published in a journal/conference before the final publication of this report. The citation is the latest (non-arXiv) version. Outside this table we cite the latest version. Venue shows the status at the time when the article was included.

AUTHOR	TITLE	VENUE	YEAR
Ahmed et al.	SYNSHINE: Improved Fixing of Syntax Errors	IEEE Trans. Softw. Eng.	2021
Austin et al.	Program Synthesis with Large Language Models	arXiv	2021
Brennan and Lesage	Exploring the Implications of OpenAI Codex on Education for Industry 4.0	SOHOMA	2022
Dakhel et al.→[61]	GitHub Copilot AI Pair Programmer: Asset or Liability?	arXiv	2022
Denny et al.	Robosourcing Educational Resources – Leveraging Large Language Models for Learnersourcing	arXiv	2022
Ernst and Bavota	AI-Driven Development Is Here: Should You Worry?	IEEE Software	2022
Finnie-Ansley et al.	The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming	ACE	2022
Gherciu	Net Impact of Large Language Models Trained on Code	Student conf.	2022
Li et al.	Competition-level Code Generation with AlphaCode	Science	2022
Puryear and Sprint	GitHub Copilot in the Classroom: Learning to Code with AI Assistance	J. Comput. Sci. Coll.	2022
Raman and Kumar	Programming Pedagogy and Assessment in the Era of AI/ML: A Position Paper	COMPUTE	2022
Sarsa et al.	Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models	ICER	2022
Vaithilingam et al.	Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models	CHI EA	2022
Zhang et al.	Repairing Bugs in Python Assignments Using Large Language Models	arXiv	2022
Al-Hossami et al.	Socratic Questioning of Novice Debuggers: A Benchmark Dataset and Preliminary Evaluations	BEA	2023
Alves and Cipriano	The Centaur Programmer - How Kasparov's Advanced Chess Spans over to the Software Development of the Future	arXiv	2023
Babe et al.	StudentEval: A Benchmark of Student-Written Prompts for Large Language Models of Code	arXiv	2023
Balse et al.	Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments	ITiCSE	2023
Barke et al.	Grounded Copilot: How Programmers Interact with Code-Generating Models	OOPSLA	2023
Becker et al.	Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation	SIGCSE TS	2023
Bellettini et al.	Davinci Goes to Bebras: A Study on the Problem Solving Ability of GPT-3	CSEDU	2023
Brusilovsky et al.	The Future of Computing Education Materials	(in draft)	2023
Bull and Kharrufa	Generative AI Assistants in Software Development Education: A vision for integrating Generative AI into educational practice, not instinctively defending against it.	IEEE Software	2023
Cipriano and Alves	GPT-3 vs Object Oriented Programming Assignments: An Experience Report	ITiCSE	2023
Denny et al.	Can We Trust AI-Generated Educational Content? Comparative Analysis of Human and AI-Generated Learning Resources	arXiv	2023
Denny et al.	Computing Education in the Era of Generative AI	arXiv	2023
Denny et al.	Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language	SIGCSE TS	2023
Denny et al.	Promptly: Using Prompt Problems to Teach Learners How to Effectively Utilize AI Code Generators	arXiv	2023
Dobslaw and Bergh→[79]	Experiences with Remote Examination Formats in Light of GPT-4	arXiv	2023
Druga and Otero	Scratch Copilot Evaluation: Assessing AI-Assisted Creative Coding for Families	arXiv	2023
Finnie-Ansley et al.	My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises	ACE	2023
French et al.	Creative Use of OpenAI in Education: Case Studies from Game Development	Multi-modal Tech. & Interaction	2023
Hellas et al.→[94]	Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests	arXiv	2023
Idialu et al.	Whodunnit: Human or AI?	-	2023
Jaipersaud et al.	Decomposed Prompting to Answer Questions on a Course Discussion Board	AI in Education	2023
Jalil et al.	ChatGPT and Software Testing Education: Promises & Perils	IEEE ICSTW	2023
Kazemitabaar et al.	Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming	CHI	2023
Kendon et al.	AI-Generated Code Not Considered Harmful	WCCCE	2023
Kiesler and Schiffner	Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments	arXiv	2023
Lau and 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	ICER	2023
Leinonen et al.→[123]	Comparing Code Explanations Created by Students and Large Language Models	arXiv	2023
Leinonen et al.	Using Large Language Models to Enhance Programming Error Messages	SIGCSE TS	2023
Liffiton et al.	CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes	arXiv	2023
Ma et al.	Is AI the Better Programming Partner? Human-Human Pair Programming vs. Human-AI pAIr Programming	arXiv	2023
MacNeil et al.	Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book	SIGCSE TS	2023
Matelsky et al.	A Large Language Model-Assisted Education Tool to Provide Feedback on Open-Ended Responses	arXiv	2023
Nam et al.	In-IDE Generation-based Information Support with a Large Language Model	arXiv	2023
Orenstrakh et al.	Detecting LLM-Generated Text in Computing Education: A Comparative Study for ChatGPT Cases	arXiv	2023
Pădurean et al.	Neural Task Synthesis for Visual Programming	arXiv	2023
Pankiewicz and Baker	Large Language Models (GPT) for Automating Feedback on Programming Assignments	arXiv	2023
Philbin	Exploring the Potential of Artificial Intelligence Program Generators in Computer Programming Education for Students	Inroads	2023
Phung et al.	Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models	arXiv	2023
Phung et al.	Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors	International J. of Management	2023
Piccolo et al.	Many Bioinformatics Programming Tasks Can Be Automated with ChatGPT	arXiv	2023
Poldrack et al.	AI-Assisted Coding: Experiments with GPT-4	arXiv	2023
Prather et al.	"It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers	TOCHI	2023
Rajabi et al.	Exploring ChatGPT's Impact on Post-Secondary Education: A Qualitative Study	WCCCE	2023
Reeves et al.	Evaluating the Performance of Code Generation Models for Solving Parsons Problems With Small Prompt Variations	ITiCSE	2023
Ross et al.	A Case Study in Engineering a Conversational Programming Assistant's Persona	ACM IUI	2023
Sandoval et al.	Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants	USENIX	2023
Savelka et al.	Can Generative Pre-Trained Transformers (GPT) Pass Assessments in Higher Education Programming Courses?	arXiv	2023
Savelka et al.	Large Language Models (GPT) Struggle to Answer Multiple-Choice Questions about Code	arXiv	2023
Savelka et al.	Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses	arXiv	2023
Singla	Evaluating ChatGPT and GPT-4 for Visual Programming	arXiv	2023
Sridhar et al.	Harnessing LLMs in Curricular Design: Using GPT-4 to Support Authoring of Learning Objectives	arXiv	2023
Wang et al.	Exploring the Role of AI Assistants in Computer Science Education: Methods, Implications, and Instructor Perspectives	VL/HCC	2023
Wermelinger	Using GitHub Copilot to Solve Simple Programming Problems	SIGCSE TS	2023
Widjojo and Treude	Addressing Compiler Errors: Stack Overflow or Large Language Models?	arXiv	2023
Yan et al.	Practical and Ethical Challenges of Large Language Models in Education: A Systematic Literature Review	arXiv	2023
Zan et al.	Large language models meet NL2Code: A survey	Annual Meeting of the ACL	2023
Zastudil et al.	Generative AI in Computing Education: Perspectives of Students and Instructors	arXiv	2023

Table 3: Venues presenting the work included in our literature review.

Venue	Count
arXiv	32
ACM	22
IEEE	5
Other Publishers	12

Table 4: LLMs and languages featured in the reviewed literature. Some papers reported on more than one (or no specific) LLM or language, so the counts do not match the number of papers reviewed.

LLM	Count	Language	Count
GPT-3/3.5	28	Python	37
Codex	20	Java	6
Copilot	12	C/C++	6
GPT-4	11	JavaScript	2
Other	7	C#	2

semantic understanding of code, such as predicting the output of a program [28] or analysis/reasoning about code [176].

2.3.2 Position papers and surveys/interviews (17): Twelve papers were surveys or position papers, summarising or discussing research conducted by others [24, 33, 51, 74, 84, 90, 109, 134, 149, 165, 195, 197]. Some of the papers focused solely on education [33, 51, 165, 195], while others included discussion on both the professional and the educational contexts [24, 84, 90]. The education-focused survey papers discuss both positive impacts—such as potentially increased productivity for both students and instructors [24, 33]—and negative impacts—such as over-reliance [33, 90]—of large language models. All papers suggest that LLMs will have substantial impact on computing education and programming more generally.

Five papers in the literature review used interviews to understand user perceptions and attitudes towards LLMs. The authors of one paper interviewed professional software developers on how they use code generation tools [52]. They found that the interviewees thought that generative AI has many use cases in software development. They note that while the tools do not require training to use, developers will need to understand the generated code for quality assurance, and to avoid over-reliance as the quality of code produced by these tools can vary. Three other papers report on interviews with students and instructors about their experiences with, and attitudes towards LLMs [121, 164, 198], finding that there is no consensus about the use of LLMs in higher education, its benefits or risks, but there is general awareness of the problem of academic integrity in the light of LLMs being used by students. One study reports on the experiences of five students using generative AI for assignments [89], highlighting both aspects of where it offered effective support, but also many limitations.

2.3.3 Studying the interactions between programmers and LLMs (9): A total of nine papers looked into interactions between programmers and LLMs. Some focused on finding interaction patterns [32, 159, 188] while others focused more on how productivity is impacted by the use of models [108, 141], whether code produced when using AI code generators is less secure than when not [173], and how students use code explanations generated by LLMs [137, 145].

Based on the findings of this research, students engage in different interaction modes when using AI code generators. These include exploration [32], acceleration [32], shepherding [159], and drifting [159]. In exploration, the programmer is unsure of what to do next, using the code generator for exploring potential approaches to tackle the problem. In acceleration, the programmer knows what they are doing and uses the LLM for producing the desired code faster. In shepherding, the programmer spends the majority of their time on guiding the LLM to produce the desired code. In drifting, the programmer drifts from one incorrect code suggestion to the next, indicating struggles in understanding the generated code.

Kazemitabaar et al. studied how novice programmer productivity and learning is affected by the use of AI code generators [108]. They found that students who used AI code generators performed significantly better (1.15x progress, 0.59x errors, 1.8x higher correctness, 0.57x time spent) without negative effects on learning. Sandoval et al. found that using code generator tools did not seem to introduce new security risks [173]. MacNeil et al. found that students generally found code explanations generated by LLMs useful for learning, but the perceived utility of the explanations and students' engagement with them varied by explanation type [137].

One study looked at how students write prompts for LLMs [71]. To this effect, the problems had to be stated in a more visually oriented way to prevent them from copying the problem statement directly, but rather write prompts on their own. Most students found the prompt writing to be beneficial, while a few voiced concerns. Another study (which we classified as assessing LLMs, i.e. category (i)) used student written prompts in order to evaluate LLMs and found it to be an effective benchmark [29].

2.3.4 Using LLMs to analyse student work (5): Five papers used LLMs to analyse student work, for example, by looking into using LLMs to fix bugs or errors in student work [20, 125, 150, 199]. Two papers looked at repairing programming errors [20, 199], one at enhancing programming error messages [125], and one at providing feedback to students based on a buggy student programs [150]. The studies that examined the performance in bug repair both reported that their results surpassed previous state-of-the-art automated program repair results. Zhang et al. reported an overall repair rate of up to 96.5% using Codex with few-shot examples and iterative prompting [199] and Ahmed et al. achieved a repair rate of 89.4% [20]. Leinonen et al. found that Codex could enhance programming error messages—which are notoriously hard for students to understand—approximately 54% of the time on average, noting that this performance is not good enough for using the model directly with students [125]. Phung et al. propose a method where instructors could balance the 'coverage' of feedback, i.e. whether a student receives feedback at all, and the 'precision' of feedback, i.e. whether the feedback is of good quality. They found that in the

Table 5: Assessment of quality metrics adapted from Hellas et al. [92].

Is there a clearly defined research question/hypothesis?	Yes: 44 No: 18 Vague / Unclear: 9
Is the research process clearly described?	Yes: 55 No: 10 Vague / Unclear: 6
Are the results presented with sufficient detail?	Yes: 57 No: 6 Vague / Unclear: 8
Are threats to validity / limitations addressed in an explicit (sub)section?	Yes, in a separate (sub)section: 38 Yes, but not in a separate (sub)section: 15 No: 18

best case, their proposed method can achieve a precision of 72.4% with a coverage of 64.2% for one of the datasets they used and a precision of 76% and a coverage of 31.2% for the other dataset.

One paper presents a tool to help students with issues without revealing the full solution [127] and reports positive feedback from both students and instructors. Other studies have also looked into mitigating risks of LLMs such as wrong answers [101] or guiding the students with hints rather than full solutions [21] (we categorised these papers as assessing LLMs, i.e. category (i)).

2.3.5 Using LLMs to generate teaching materials (5): Five papers looked into using LLMs to generate teaching materials [69, 75, 139, 174, 186]. In two cases, the teaching materials being generated were programming exercises [75, 174]. One of the main findings in both papers was that LLMs can be coaxed into generating exercises with prescribed themes (such as basketball or cooking) and programming concepts (such as loops or conditionals). In addition, the exercises generated by LLMs were novel and sensible, although the authors cautioned that the quality might not be good enough to provide the LLM-generated exercises directly to students. Another study that compared LLM-generated content with student-generated content concluded that the quality is comparable, but still recommends further research [69]. Similarly, LLMs were found to be able to generate reasonable learning objectives [186]. One paper presents a new tool, but does not offer an actual evaluation of it [139]. All papers suggest that using LLMs could help instructors save time in generating teaching materials.

2.4 Impact on teaching and learning

The broader literature on LLMs and their potential effects is often organised around the dichotomy of opportunities and risks [46, 107, 162, 185]. For example, Bommasani et al. produced an extensive report documenting the opportunities and risks of foundation models across a broad variety of domains, including education [46]. Kasneci et al. documented similar opportunities, risks and mitigation strategies, specifically focusing on the use of ChatGPT in education [107].

Among the papers in our dataset, there was broad agreement that LLMs would have a major impact on teaching and learning in computing courses. Authors identified various opportunities and risks for both students and teachers and we present these in the following sections.

2.5 Opportunities

The papers we reviewed identified a number of potential opportunities that could positively impact computing education. One of the prominent opportunities that emerged was related to reducing instructor workload, for example by generating large banks of diverse learning resources and support materials [69, 137, 174], automating various aspects of the grading process [195], and providing personalised help to students who are struggling and who would otherwise consume considerable instructor effort [21, 51, 145]. Related to this theme, Bull and Kharrufa argue that the type of scaffolding that AI tools can provide can "support the student in their learning and ... offload some of that ... burden from the educator" [52].

Improving the learning experience for students was another common opportunity that emerged. Several papers described the idea of using an LLM as an assistant or pair programmer, which represents a significant change from current pedagogical practice [123, 152, 169]. As a concrete example of the kind of assistance that could be provided while students are programming, Leinonen et al. suggest that LLMs could help students understand terse error messages [125] which have traditionally been a source of difficulty for novice learners [36]. Several groups also identified opportunities for creating new tools around LLMs, e.g., to support repair of syntactically incorrect code [20], to help answer questions [94, 108] or provide hints [145], or even to support the crowdsourcing of new questions [75]. Indeed, some recent papers found in the second phase snowballing focused on introducing tools around LLMs, such as CodeHelp [127] and Promptly [71]. Despite the promise of using AI tools for learning support, Dakhel et al. and Prather et al. caution that although they can be a great asset for professional developers, they may be less helpful for novices if the tools generate non-optimal or erroneous outputs which could cause confusion [61, 159]. As the quality and performance of the models improve, this may be less of an issue as time goes by.

Another recurring opportunity mentioned in the papers we reviewed was the potential for a renewed focus on problem solving.

For example, Vaithilingam et al. explored the usability of code generation tools and suggest they can be used to rapidly provide a good starting point for a solution, thus allowing programmers to focus on the problem solving process and reducing the need for a focus on lower-level details [188]. In a similar vein, Denny et al. [70] and Prather et al. [159] explore the use of Copilot in two different contexts, suggesting it can be used to teach students how to express problem solutions in natural language, and to focus on guiding students through problem-solving strategies, respectively.

Finally, LLMs present clear opportunities for instructors to rethink assessment practices and reconsider what assessment means in computing courses [39, 70, 153, 176]. Raman et al. suggest assessments could focus more on code understanding, such as tracing and verification, and less on syntax and code writing [165]. LLMs can also be used to generate a variety of flawed solutions, providing plentiful opportunities for incorporating code review tasks [85]. The systematic literature review of Yan et al. explored practical and ethical challenges of LLMs in education, categorising some of this prior work around the 'Assessment and grading' category and argue that grading student assessments is a promising application of LLMs [195]. The impressive documented performance of large language models in solving typical CS1 and CS2 problems suggests that some rethinking of assessments is essential [86].

In summary, the papers we reviewed suggest that LLMs present a wide array of opportunities for computing education such as improving instructor productivity by reducing workload, enhancing student learning experiences, enabling a greater emphasis on problem-solving, and suggesting new assessment practices.

2.6 Risks

Several risks were identified by the papers we reviewed. Authors were concerned that generative AI could be used in ways that limit student learning or make the work of educators more difficult [90].

2.6.1 Risks for students. The learning resources produced by generative AI pose significant risks to student success. Wermelinger [191] and Sarsa et al. [174] observe that explanations of code can be a useful learning resource, but if the explanations contain mistakes then learning could be negatively impacted. Since AI generated content is presented authoritatively (and is frequently correct), students are unlikely to question the content and may learn incorrect information [52, 103]. Automatically generated tests may be partially complete, leading students to inadequately test their code [191]. The resources created by LLMs might also have less variation than those created by humans [69] and thus limit the variety of examples to which students are exposed. AI generated content is not curated for specific courses, so learning material generated could potentially include syntax, programming constructs, or other content that is inappropriate for students in a given course [33, 94]. Example programs used by students for learning could have poor implementation or poor style, which may result in students acquiring undesirable programming habits [85].

The use of generative AI may also result in students spending time in unproductive ways. Wermelinger [191] speculated that students may spend excessive time on prompt engineering in the hope of hitting on a successful solution rather than making progress towards the solution, which was indeed observed in the study by Prather et al. [159] in the 'shepherding' interaction pattern. Bull and Kharrufa [52] suggested that it may take longer to figure out an effective prompt than to write a solution. Hellas et al. [94] found that LLMs tended to hallucinate issues in student code which could cause students to focus on these non-existent issues instead of the actual issues in their code.

Wermelinger [191] observed that explanations generated by Copilot focused on a line-by-line description of *what* the code did rather than how it achieved the goal desired. This is supported by the findings of Sarsa et al. [174] who noted that Codex seems most proficient at crafting line-by-line code explanations, as opposed to higher level summaries of the code. This is akin to a multi-structural explanation [180], which may focus student attention at lower levels of the SOLO taxonomy, rather than thinking about the overall purpose of code. However, non-code models such as GPT-3 have been found to be more apt at creating higher level code explanations [137].

The most common concern expressed by authors about student learning was the potential for students to become over-reliant on generative AI tools to solve problems [33, 85, 123, 126, 169] and assist in debugging code [125, 150, 199]. Students who rely on generative AI may be misled into believing they are making progress, and this illusion of capability may reduce their self-understanding about their level of mastery of the subject matter [158, 159].

As introductory students realise that generative AI can outperform them on most tasks, they may lose motivation to learn the material, and become demoralised about the future of computing [123]. Further, novice learners of programming may become overwhelmed and confused by generated code, which could add to the high levels of frustration that are common in introductory programming courses [71, 188].

2.6.2 Risks for teachers. Several authors raise concerns about the impact of generative AI on teachers and teaching practice. Unsurprisingly, issues of academic integrity were the most common concern raised.

Generative AI is reported to perform very well in assessments that are commonly used in introductory courses, raising concerns that students will submit solutions that they have not created themselves [33, 70, 85, 86, 121, 152, 169, 176, 177]. The solutions generated by AI might not be easily identified as AI-generated [160], which requires teachers to adapt and develop new teaching strategies to ensure academic integrity is maintained [191].

Wermelinger [191] recommended that educators stop 're-dressing' toy problems because generative AI will provide good solutions which will restrict the learning opportunities for coding, debugging and algorithmic thinking, compared to problems with interesting 'wrinkles'. Teachers who shift away from using many small problems to create larger and more authentic problems in an attempt to reduce reliance of generative AI will lose access to the quick and easy assessment methods such as automated grading associated with many introductory programming courses [86]. Educators will need to develop new resources to explicitly address LLMs and guide students instead of leaving them alone with the tool [188].

Teachers who use generative AI to assist in the creation of learning resources may unintentionally produce exercises that are underspecified or that contain incorrect reference solutions or inadequate/incorrect test cases [174]. Teachers who are concerned about the impact of generative AI may intentionally modify course delivery in ways that reduce the effectiveness of their teaching practice (e.g., by increasing academic integrity at the cost of scaffolded programming exercises), or adjust the curriculum to shift focus away from code writing, leaving students poorly prepared for subsequent programming courses [165].

Although there is a growing need to teach students how to use generative AI appropriately, it is unclear how we should do so. Barke et al. [32] discuss the need to balance the introduction of generative AI too early in the curriculum where over-reliance is a possibility, against introducing generative AI too late and failing to provide an authentic experience relevant to changing landscapes including industry practice. Bull and Kharrufa [52] note that it is challenging for novices to understand generative AI capability and use prompts effectively, suggesting a need to formally teach students to effectively use the tools they have at their disposal.

2.6.3 Risks for the community and society at large. The use of generative AI also raises concerns for the broader community. As more code is likely to be generated automatically, there is potential for biases to be unintentionally introduced due to inherent bias in models and because of bias in the training data [33]. Generated code may contain security vulnerabilities and bugs [33, 52, 173]. Also very concerning, Kazemitabaar et al. [108] found that students with more knowledge benefited more from code generation than students with less knowledge. Similarly, Nam et al. [141] found that professionals benefited more from AI code generation than students. These findings suggest that AI code generators may widen the gap between over- and under-achievers, exacerbating teaching challenges arising from classes with heterogeneous ability levels. In response to these issues, Prather et al. discuss design considerations for generative AI tools that could potentially mitigate these risks and lead to more direct benefit for novice programmers [159]. Despite a dearth of empirical work to-date, it must be noted that Generative AI may serve to exacerbate already present issues in computing. For instance, Generative AI may favour some groups while disadvantaging minoritised groups. Although, how Generative AI will actually influence such issues remains to be seen, and it is possible that it may also be used for positive effect in the future.

2.7 Limitations and threats to validity

Our literature review was conducted from April to August 2023. Therefore work published subsequently is not included. Due to the fast pace of work on Generative AI in all disciplines including computing education, only literature published in less than a three year period (2021 to August 2023) was considered. For this reason we conducted a single round of snowballing (i.e., we did not do subsequent snowballing on the papers found in the first round of snowballing). This may have omitted some work that failed to reference the most visible early work (our "seed" papers), but we do not believe that this will include significant numbers of papers or change the general trends and themes identified in our literature analysis.

The inclusion of results from arXiv and other literature sources is driven by pragmatism. The machine learning community makes wide use of arXiv due to the fast-paced nature of the field, and if we omitted it, we would miss the most recent results in an already narrow window of time. However, the inclusion of these sources admits work that has not yet undergone peer review. During our deep review of the included papers, there were a few concerns about the quality of a given paper. We retained these papers as they met the inclusion criteria, and note that, on the whole, the papers appear to be ready for review and demonstrate many of the criteria for quality proposed by Hellas et al. [92].

3 SURVEY OF STUDENT AND INSTRUCTOR PERCEPTIONS ABOUT GENAI

Students and instructors may have quite different views of the use of generative AI tools in computing classrooms. For example, one of the well-documented concerns regarding generative AI in educational contexts is that students may become over-reliant on them for the generation of answers [74, 198]. In this case, students who rely on the tools may initially take a more positive view of them when compared with instructors, however, their views may change over time. Given the speed with which generative AI tools are being developed and adopted, documenting student and instructor perceptions at the current time provides a useful snapshot of current practice and sentiment and is aimed at facilitating future explorations of how views may change as these tools become more embedded in educational contexts.

In this section, we report the findings from two surveys, one with computing instructors and the other with students, which we conducted from July to August 2023 with responses spanning 20 countries. We first review similar explorations in computing and other disciplines, and then after describing our methods we organise our findings around insights derived from analysis of both quantitative and qualitative data.

3.1 Prior explorations of student and instructor perceptions

Several recent studies have explored the perceptions of students and teachers toward the potential impact of generative AI in broad educational settings. Chan and Lee acknowledge a generation gap in how generative AI is perceived [54]. Using an online survey involving 399 students and 184 teachers predominantly from Hong Kong but across a diverse range of academic disciplines. They examine distinctions in perceptions, experience, and knowledge of generative AI between educators and students across different generations, classified as Gen Z (students) or Gen X and Y (teachers). They observe that while students are generally optimistic about the use of these new technologies, teachers hold more concerns regarding over-reliance and ethical issues, and were also more sceptical about the abilities of generative AI tools. They emphasise the urgent need for clear policies and guidelines to ensure that academic integrity is maintained and to promote equitable learning experiences. In follow-up work, Chan addresses this need by proposing an AI policy framework specifically for higher education [53]. This encompasses three dimensions: pedagogical, which uses AI to enhance teaching

and learning outcomes; governance, which addresses privacy, security and accountability issues; and operational, which pertains to infrastructure and training. To inform the policy, they conducted an online survey of 457 students and 180 teachers and academic staff from Hong Kong universities. They argue that the student voice plays an essential role in the drafting and implementation of policy. In general, both students and teachers reported limited experience with AI tools, suggesting potential for growth in adoption and the need for training on the effective use of AI technologies. This could also indicate that policies being drafted now may need to change in the future as experiences, practices, and attitudes change.

In a related strand of work, Chan and Tsi focused specifically on the capacity of generative AI for replacing human teachers [55]. Their rationale for this direct question was to assist educators in preparing for the inevitable integration of AI into educational settings. The authors review existing literature on the role of AI in the classroom and present a synthesis of its limitations, classifying these into eight categories covering 26 aspects. For example, the category 'Emotional and Interpersonal Skills' highlights the social-emotional competencies of human teachers and covers aspects such as human connection, cultural sensitivity and building trust and rapport. An online survey consisting of 11 closed items and several open-response questions was distributed to universities in Hong Kong and received responses from 144 teachers and 384 students. Students generally indicated an appreciation for the unique emotional qualities of human teachers, whereas they expressed concern about student misuse of generative AI tools. Despite some variation in responses, both students and teachers generally agreed that AI is not likely to entirely replace human teachers and in particular the social-emotional competencies only they can demonstrate.

A recent study by Amani used a survey to measure student and instructor perceptions of generative AI in academia with the goal of capturing perceptions, misconceptions, concerns, and current usage trends [25]. They argue that it is essential to report instructor and student perceptions now given the rapid changes and improvements in the tools that are underway. Two online surveys were created: a student-oriented survey focusing on current usage and perceptions; and an instructor-oriented survey focusing on how generative AI is affecting their current courses and how they think students should use it. Data was collected from 243 staff and 813 students at Texas A&M university, revealing a clear perception that resisting these new technologies is likely not feasible, and that teaching practices must adapt in response. Students value the high availability of the tools, but recognise the potential for their misuse. Forman conducted a similar online survey exploring student perceptions of ChatGPT [88]. Analysis of 71 responses to the 7-question survey revealed that students generally had a positive long-term view of the role that such technologies would play in their lives, and that they currently relied on ChatGPT to save time when working on assignments and projects.

Raman et al. investigate the factors that influence the adoption by university students of ChatGPT [166]. Their work, which utilises Rogers' Diffusion of Innovation theory as a conceptual framework, proposes that five attributes of the technology influence its adoption, namely relative advantage, compatibility, ease of use, observability, and trialability. Their empirical analysis, which is based on a survey of 288 students, supports their hypotheses and indicates gender-based differences in how students prioritise the attributes.

Although online surveys have been a popular instrument in work exploring student and instructor perspectives, a few recent interview studies [121, 164, 190, 198] have investigated the impacts of generative AI on computing education research and practice. Notably, Lau and Guo recently conducted in-depth interviews with instructors to understand how they planned to adapt to the emergence of tools like ChatGPT and Copilot [121]. They conducted Zoom interviews with 20 instructors from nine countries. The interviews were framed around a hypothetical question, where participants were asked to imagine a future where students had access to an AI tool that could both write perfect code for any programming problem and that was undetectable to plagiarism detection methods. Instructors were asked to describe how they would adapt their pedagogical approaches over the short- and long-terms. In the short-term, the primary concerns centred around cheating and plagiarism, to which instructors have responded by relying more heavily on invigilated exams and educating students about current model limitations. Longer term perspectives varied, with one school of thought aiming to resist AI tools and teaching in conventional ways, and the other aiming to integrate AI tools into the curriculum to better prepare students for the changing requirements of industry. Specific examples from this latter category included using AI to provide more personalised help to students, using assignments that focus more on code reading, critique, and more open-ended design, in addition to using AI to evaluate new types of assessment tasks. These instructors also viewed AI as being potentially useful for broadening participation and accessibility in computing due to their capacity for providing personalised assistance. Significantly, whether they tended towards resisting or embracing AI tools, instructors generally agreed that the objectives of computing education will likely need to change to adapt to the growing influence of AI.

Zastudil et al. [198] conducted Zoom interviews with six CS instructors and 12 CS students. The analysis compared and contrasted their experiences, hopes, and concerns about the emergence of generative AI in computing education. Students and instructors aligned on key concerns such as over-reliance, model trustworthiness, and plagiarism; however, they diverged regarding how each group preferred those issues to be addressed. Students stressed the importance of crafting engaging and culturally relevant assignments as well as reducing busy work to address plagiarism, whereas instructors proposed increasing the weight of proctored exams. Students were concerned about the quality of the generative AI responses, and instructors were concerned that students would be unable to identify incorrect or misleading responses. Instructors and students were both excited about the potential for GenAI tools to shift course topics toward higher-levels of abstraction, such as design patterns.

Wang et al. [190] conducted a three-part study which culminated in Zoom interviews with 11 instructors. The authors found that instructors are concerned that students will misuse or over-rely on generative AI tools, but instructors did not have plans to adapt their courses due to a current lack of effective strategies. Instructors believed these problems would be harder to address in introductory courses. Similar to the findings of Zastudil et al. [198], instructors

were concerned about how incorrect model responses might lead students to develop faulty mental models.

Rajabi et al. [164] interviewed 36 instructors in-person and 4 instructors virtually. Their interviews uncovered four primary themes that related to adapting pedagogy, plagiarism, assessment, and job preparedness. Instructors raised concerns about the trustworthiness of generative AI tools and their capacity to mislead students. However, instructors also argued that generative AI tools should not be banned because students will continue to find ways to use them. Instructors advocated for completing in-class assignments to mitigate plagiarism concerns, but acknowledged that this could increase student anxiety about exams and grade weight—a concern that has been previously raised in computing education [110, 120, 136].

Based on these prior interview studies, the goal of our survey was to provide a large-scale, systematic, and international overview of the experiences students and instructors have had with generative AI in computing education contexts and to uncover their preferences for how these models should be used in computing classrooms.

3.2 Methods for data collection and analysis

To better understand the perceptions and experiences of students and instructors in computing courses as they relate to Generative AI tools, we developed two surveys—one for students and a second for instructors. We designed the survey to have questions that were asked to both groups to facilitate comparisons between these two crucial stakeholders. This method also draws inspiration from previous studies that directly compare the responses from students and instructors to the same questions [53–55]. We also draw inspiration from previous large-scale surveys in computing education research. The use of online surveys and recruitment of participants via bulk email such as the SIGCSE mailing list is a common method, and has been used in work by Denny et al. [67], Schulte and Bennedsen [178], Elarde and Fatt-Fei [83], and Dale [63]. The following sections describe how participants were recruited and how the survey was constructed.

3.2.1 Recruitment and participants. We recruited 57 instructors and 171 students to complete the corresponding online surveys.

To recruit instructors we utilised the mailing lists of computing education professional groups including: ACM SIGCSE; the Australasian ACM SIGCSE Chapter; the UK ACM SIGCSE Chapter; the Ireland ACM SIGCSE Chapter; and the China ACM SIGCSE Chapter WeChat group. The goal for targeting these mailing lists was to draw a broad sample of computing education practitioners and researchers. However, we recognise that the resulting sample likely results in a selection bias of instructors who are particularly invested in computing education compared with their peers. This is a well-known challenge, as noted by Schulte and Bennedsen [178]. In an attempt to address this, we included a request for them to share the recruitment materials with colleagues in their department. This snowball sampling technique was also used by ITiCSE working group members to share the recruitment materials in their personal networks. Still, this approach does risk over-representing regions that have SIGCSE Chapters.

To recruit students we also used a snowball sampling method where instructors were requested to share a recruitment announcement with students through their courses and departmental mailing lists. In this case, it is possible that we introduce response biases due to factors such as as high-achieving students being more likely to respond to the survey. To address this potential we included the phrase "if you have struggled with your computing courses, your voice is especially appreciated to ensure better experiences for students like you in the future".

3.2.2 Survey design. We developed surveys to focus on critical topics that have recently emerged across birds-of-a-feather discussions [135], workshops [138], and position papers [33, 74]. These topics include calls for curricular and pedagogical changes, consideration of ethics, and a need for replications and benchmarking. We also included questions inspired from work on plagiarism in programming assessments [22], student help-seeking behaviour [80], and from the related work discussed in Section 3.1.

This resulted in 35 questions for the student survey and 42 questions for the instructor survey (counting all open- and closed-response questions). The overlap between the student and instructor surveys included 27 questions that were either identical or differed due to minor rewordings that aimed to improve the readability between groups (e.g. "... using GenAI tools in ways that your instructors would not approve of?" on the student survey was reworded as "... using GenAI tools in ways that you would not approve of?" for the instructors).

The questions used in the student survey are listed in Appendix B, and those in the instructor survey questions are listed in Appendix C.

3.2.3 Thematic analysis. To analyse responses from students and instructors for the open-response questions we followed an approach for thematic analysis similar to the reflexive process described by Braun and Clarke [58]. The reflexive thematic analysis process is not prescriptive, but provides guidance for the phases needed to robustly explore, interpret and report patterns in qualitative data. Given that the resulting dataset was not very large (a total of nine open-response questions in common on the instructor and student surveys, and a total of 228 responses across both groups) the questions were divided between two researchers who analysed all responses to the questions they were assigned.

Each researcher began by reading the responses to familiarise themselves with the data, and then defining succinct labels that were assigned to each response that captured important features of the data. Practically, this process used a spreadsheet in which responses were listed in rows and the labels that were defined for coding the data were listed in columns. The final steps of the analysis involved grouping the labels into broader themes suitable for reporting.

3.3 Quantitative insights

3.3.1 Demographics. We recruited 57 instructors from 12 countries with an average of 18.2 years of teaching experience. Participants lived primarily in the USA (45.6%) with the United Kingdom (17.5%), Canada (8.8%), Jordan (5.3%), and Pakistan (5.3%) rounding out the top five countries. The most common class sizes that instructors

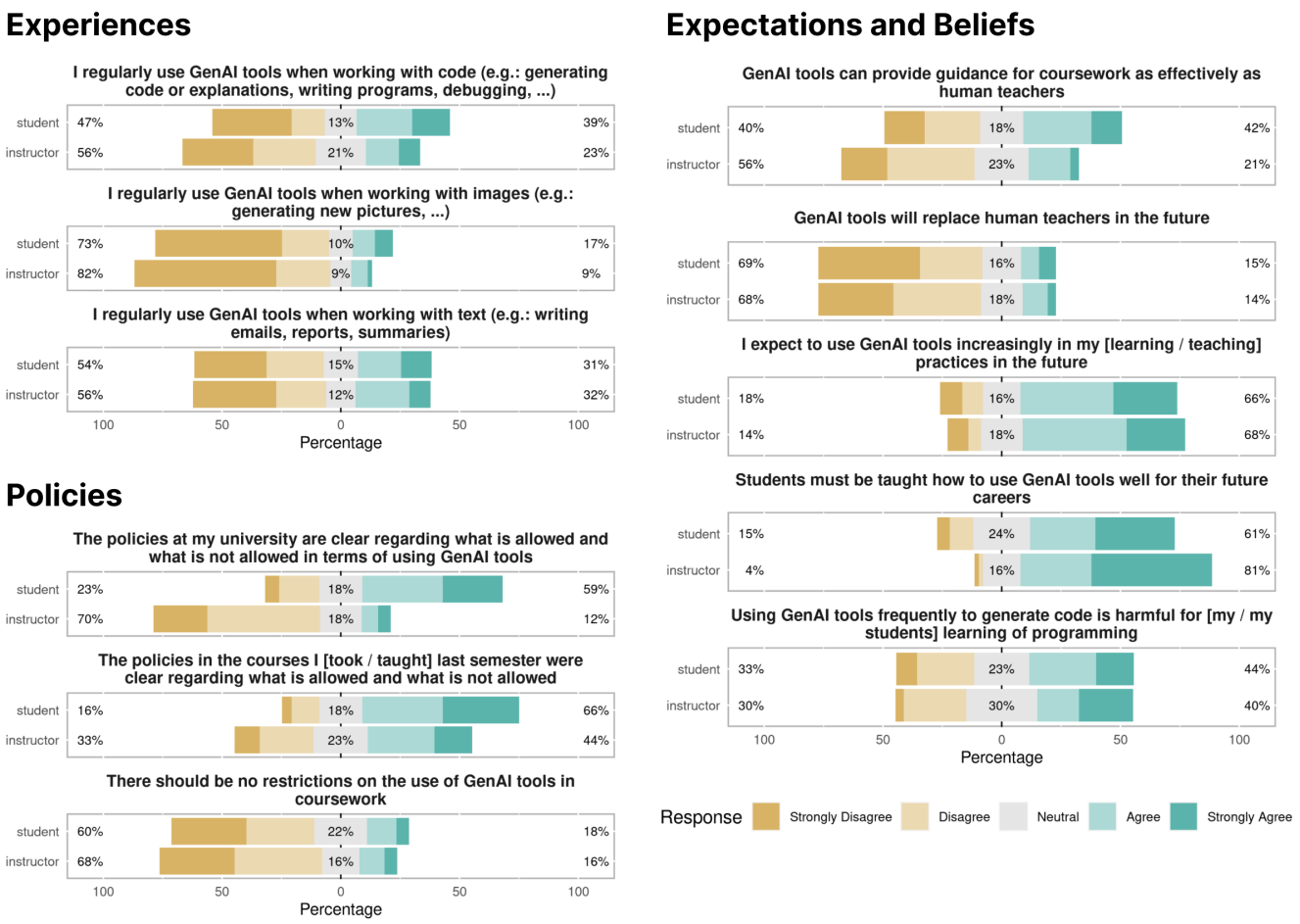


Figure 2: Summaries of the survey responses from 171 students and 57 instructors: 1) Students' and instructors' perspectives were compared along Likert scale responses. The displayed percentages show the fraction of respondents with negative (i.e.: strongly disagree or disagree), neutral, and positive (i.e.: strongly agree or agree) responses.

reported teaching were in the 11-30 (36.8%), 31-50 (26.3%), or 100-250 (24.6%) ranges. The majority of instructors self-identified as men (77.2%) with far fewer instructors identifying as women (19.3%) or non-binary (3.5%).

Through our snowball sampling method, we additionally recruited 171 student participants across 17 countries. The top five countries included New Zealand (35.7%), Jordan (17.5%), the USA (14.0%), Indonesia (8.8%), and Australia (7%). About half of the students self-reported being in their first year (48.5%). Students in their second, third, and fourth years accounted for 21.6%, 19.9%, and 7% of the respondents, respectively. The average number of courses taken was 4.6 with 38% of students only having taken one course. 90% of students had taken 10 or fewer courses. Most participants selected computer science as their major (42.7%). Additional majors included undeclared engineering (15.8%), software engineering (12.3%), computer engineering (7.6%), data science (5.3%), and information technology (4.1%). There were five participants who majored in either chemistry, supply chain, mathematics, economics, or psychology; and two students majored in physics.

3.3.2 Comparison of student and instructor perceptions. Figures 2 and 3 summarise students' and instructors' responses to the Likert scale questions on the survey. Responses from students and instructors to questions related to experiences and expectations were largely aligned. However, some important differences emerged for questions focusing on course policies. In this subsection, we review the results and briefly discuss the implications.

Experience and usage. Students and instructors shared similar experiences with GenAI tools, using them primarily for writing code and working with text. However, fewer individuals in both groups used GenAI tools for tasks involving images. Students had slightly more experience using GenAI for writing code than instructors. While the difference is currently minor, instructors should keep in mind that students may rapidly become more expert at using GenAI tools. In light of this possibility, instructors should proactively stay informed about these tools' capabilities, even if they do not intend to incorporate them into their courses. This proactive approach is

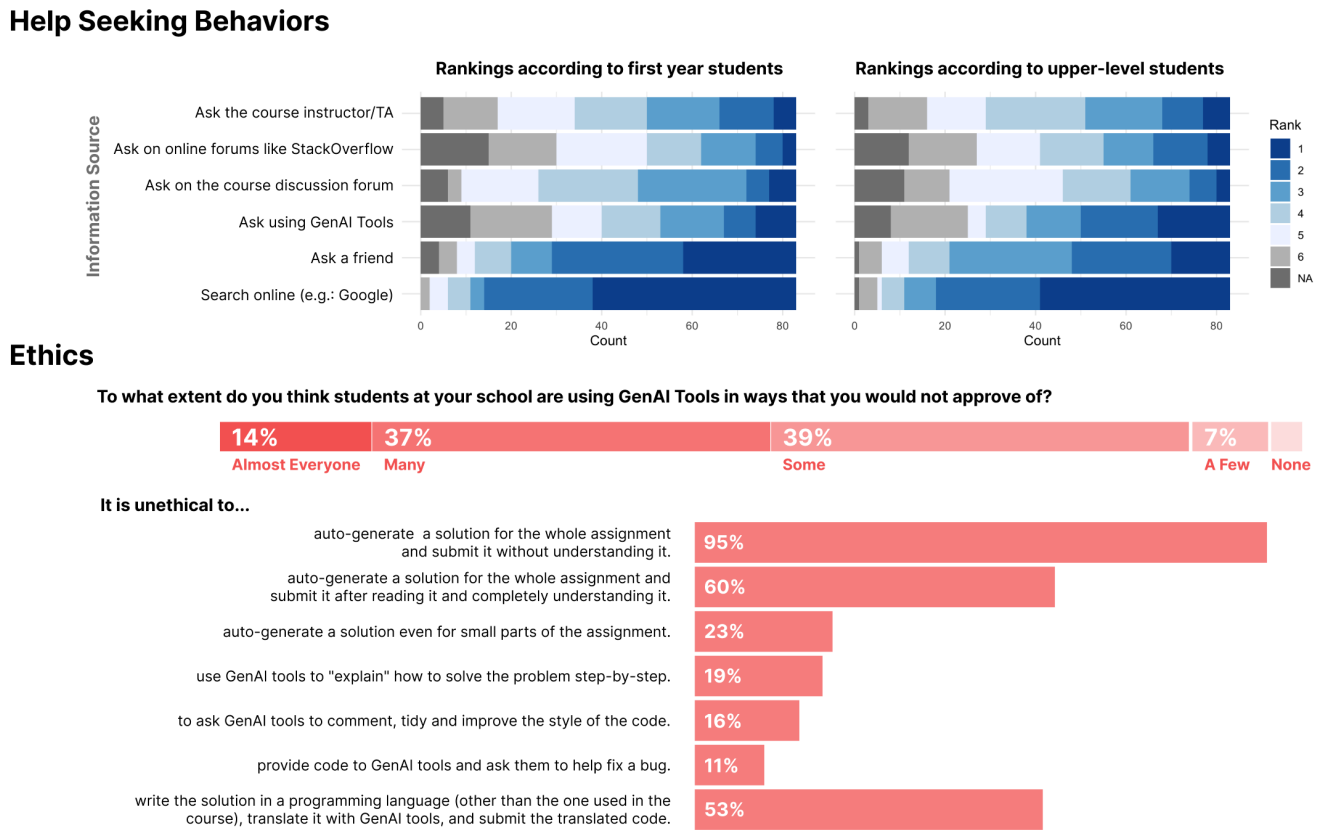


Figure 3: Summaries of the survey responses from 171 students and 57 instructors: 1) students ranked their help seeking preferences from 1 to 6, and 2) instructors shared their beliefs about the ethical use of Generative AI Tools.

crucial to ensure instructors can continue providing meaningful educational experiences to students and remain well-informed about evolving technological advancements.

> **Similar use, different experience levels:** Students and instructors use GenAI similarly, for writing code and working with text. However students had slightly more experience using GenAI for writing code. Instructors should keep abreast of developments even if they do not plan on incorporating Generative AI in their courses.

Course and institutional policies. Students and instructors were aligned in their belief that some restrictions should be placed on the use of GenAI tools in their coursework. However, there may be some misalignment around what those restrictions should be. While students had mixed opinions about whether the university policies were clear, instructors largely disagreed with the statement that the policies were clear. This misalignment could lead to challenges where students are following implied policies rather than explicit policy guidelines. Given the shared belief that use should be limited, it is important that students, instructors, and institutions be aligned on course and institutional policies. However, it should be noted that students and instructors agreed slightly more strongly that course policies were clearly defined.

> **Policy and its application pose challenges:** Students and instructors feel that some restrictions should be placed on the use of GenAI tools in coursework but differ more in what those restrictions should be and when they should apply.

Expectations and beliefs. Based on the responses from students and instructors, there was a close alignment in expectations and beliefs regarding GenAI tools. Both groups strongly agreed that GenAI tools cannot replace human instructors and that human teachers provide more effective guidance than GenAI tools. However, both students and instructors also expected GenAI tools to play an increasing role in the future of their teaching and learning, as well as in students' future careers. This suggests that while GenAI tools do not currently replace the value provided by instructors, it is important for instructors to reflect on and clearly define their value to students. It may be that students rely less on instructors for explanations and help, but rely on them more for curating the

learning environment and ensuring that learning objectives are being achieved.

> **Human instructors are essential, but GenAI is here to stay:** Students and instructors agree that GenAI tools cannot replace human instructors, and that human teachers provide more effective guidance than GenAI tools. However, both groups expected GenAI tools to play an increasing role in the future of their teaching and learning, and in students' careers.

3.3.3 Help-seeking behaviours. We surveyed students about their help-seeking behaviours to understand how prominently GenAI tools are being used by students when they require assistance. The results indicate that students predominantly continue to favour web searches as their primary resource for help. Nevertheless, GenAI tools are progressively establishing themselves as a dependable resource, surpassing online discussion forums as a preferred source of help. Interestingly, the extent of students' reliance on GenAI tools appears to be influenced by their academic program stage. Upper-level students exhibit a greater tendency to use generative AI tools over other resources, including peers, instructors, and teaching assistants. In contrast, first-year students still exhibit a preference for seeking assistance from their peers when facing challenges. This may reflect differences in the kinds of help students are seeking at different stages in their academic program or differences in their willingness to adopt new technologies, such as GenAI tools.

> **Using GenAI for assistance is in flux and varies by student level:** The role of GenAI for help-seeking seems to be affecting forums more than web searches, and GenAI is used more by upper-level students while lower-level students leverage peers more.

3.3.4 Ethical use of Generative AI tools. To better understand the ethical uses of GenAI tools, we surveyed instructors about the scenarios of use that they considered unethical. The findings reveal a consensus among instructors that auto-generating an entire assignment solution is considered unethical when students lack comprehension of the generated code. However, instructors held differing opinions on the ethics of generating solutions for an entire assignment when students possess a full understanding of the generated code or when students write code in a different programming language and then translate it into the language used in the course. In these cases, approximately half of the instructors deemed such practices ethical, while the other half considered them unethical. This suggests that instructors may be supportive of students using these tools as long as students demonstrate a clear understanding of the task and achieve the intended learning outcomes of the course. Along this line of reasoning, instructors generally concurred that it is acceptable for students to employ these tools to generate solutions for specific portions of assignments, facilitate code debugging, elucidate concepts, or enhance the style and readability of their code. These are situations where the tool could save time without negatively affecting learning outcomes. Finally, when asked about the extent to which instructors believed that their students were using GenAI tools unethically, around half (50.8%) believed that many or almost all of their students were using the tools unethically.

> **What constitutes ethical use is complex:** What is considered ethical varied between instructors and students. Instructors should provide clear guidance in their context.

3.4 Qualitative insights

3.4.1 Instructor use of GenAI. We asked instructors to describe the ways that they currently make use of GenAI tools, seeking separate responses for text generation and code generation (Questions 16 and 17 in Appendix C). Overwhelmingly, for both types of content, the most common response from instructors was that they were not currently using GenAI tools. Half of instructors reported they had not used GenAI tools for text generation and 40% said they had not used such tools for code generation. A small number of these instructors (two and four for text and code generation, respectively) indicated they planned to use GenAI tools in the near future. For example, one instructor planned to use text generation tools to aid in preparing drafts for problems (i.e. *"None currently, but plan to use in the near future for generating ungraded practice problems or first drafts of graded problems"*), and another instructor planned to start using code generating tools in the upcoming semester (i.e. *"So far I have not, but I will next spring to help write code."*). This points to an emergent interest in GenAI from instructors in our survey and recognition of the use of GenAI for teaching.

The primary theme to emerge from responses around the current usage of text generation tools was for the creation of a wide variety of learning resources. Of these resources, equally popular was the production of assessment questions (i.e. *"Occasionally will work with ChatGPT to ideate exam questions"*) and for aiding in various kinds of writing tasks such as report writing and turning brief notes into longer form prose (i.e. *"Generate readable sentences of my brief notes"*). Other types of text-based artefacts that instructors reported creating were course materials, examples for students, explanations of complex algorithms, and scenarios for highlighting ethical issues in software engineering.

Several other interesting uses of text generation tools were mentioned. Several instructors described using such tools to support other tasks, such as for performing background research, overcoming writer's block, and for paraphrasing papers when constructing references (i.e. *"creating a reference for paper and paraphrase"*). Several instructors highlighted the summarisation capacity of GenAI tools since they can effectively condense long-form text content. One instructor used this feature to extract insights from written student feedback (i.e. *"paste in student feedback about course and ask GenAI to summarise for me"*). Finally, one instructor reported integrating text generation capabilities into other tools designed to

support students (i.e. *"We are actively building a tool to help respond to common questions for students in forums"*).

> **Low uptake of GenAI tools:** The survey revealed that most instructors are not currently using GenAI tools for text or code generation, but some have expressed plans to integrate them in the future. The tools, when utilised, are primarily used for creating diverse educational materials, however, satisfaction regarding the quality of outputs vary.

When reporting their use of code generating tools, instructors describe a variety of tasks that involve creating code in varying levels of detail. Many responses to this question described fairly generic use of such tools (e.g. *"Code writing"*, *"generating part or some of function"*), whereas some were much more specific. For example, several instructors described generating code examples that they would then give to their students to modify or analyse. One instructor described generating code as a way to help them understand the suitability of certain topics and common coding patterns (i.e. *"I use GenAI tools to write initial code on topics I am looking into including in coursework or learning more about for course purposes in order to understand common forms of code"*).

Another reported use was for generating programming exercises that could be given to students for practice. This included some novel ideas, such as asking students to compare their own code with code generated by the AI tool, and asking students to use ChatGPT to generate code and then critique the output that it produces, including highlighting necessary changes. One instructor noted that attempts to generate exercises suitable for their course were largely unsuccessful due to lack of context regarding the course structure (i.e. *"But because it lacks context about the ordering of course concepts and the goals of the exercises, it has not been much help"*). Another instructor also mentioned that such tools were not particularly helpful to them for coding, noting that they often found it quicker to write the code themselves, but that they did find value in using it to generate data (i.e. *"I used it fairly heavily in a database course to generate sample data"*).

Overall, most instructors who participated in our survey were not currently using GenAI tools, although several were explicit in their plans to do so in the near future. Those that were using them were doing so to generate a broad array of educational content, including assessment questions, practice exercises and examples, although not all appeared satisfied with the outcomes.

3.4.2 Instructor observations of student use of GenAI. We asked instructors to describe their observations regarding how students are currently using GenAI tools (Question 26 in Appendix C). The most common response to this question, mirroring the earlier results regarding their own use of the tools, was that they had not observed students using GenAI tools. However, this was relatively less common (reported by fewer than one-third of participants), suggesting that instructors have observed their students using GenAI tools more than the instructors use the tools themselves.

The next most common theme that emerged, appearing in 20% of responses, was around academic misconduct. Instructor responses for this theme indicated that students frequently use generative AI tools to cheat on their assignments, in-class exercises, projects and on exams. They noted students using AI for generating complete solutions, including "blindly copying and pasting solutions", and submitting these as their own work even when they sometimes contained advanced elements that were not taught in the course. One instructor responded to the question about how their students are using GenAI tools with: *"Comprehensively. They are feeding my assignments into ChatGPT and directly copying results and handing them in"*. This misuse of the tools was a clear concern for instructors, and highlighted problems around over-reliance (i.e. *"they don't check and don't understand the solution generated most of the time"*, and *"they don't realise that it generates something very different from what was asked"*).

More positive uses of the tools were also reported. The next most common theme was around using GenAI tools to debug and understand code. Instructors reported observing students use AI for debugging purposes (i.e. *"they have used it to help fix errors and better understand compiler messages"*), to generate test data and code (i.e. *"writing test cases or code to generate test data"*), and for explaining code that they do not understand. A similar number of responses also focused on generic code writing help, such as *"to complete small coding exercises"*. Two instructors mentioned that the students they had observed using GenAI for writing code actually found the experience frustrating, noting that it would have been easier to write the code themselves. A related, but less common theme, was around the use of GenAI tools as a conceptual learning aid. A few instructors discussed students using AI to help them understand topics better from the class, and assisting with ideation for project work but not giving complete solutions (i.e. *"They are using them to better understand topics from class (when they miss a meeting, get distracted, whatever)"*).

An interesting theme emerged around language enhancement and communication. Several instructors observed students using generative AI tools to help improve their English language skills, both in their essays and in communicating with others online, such as writing emails or making posts on forums (i.e. *"We have a variety of students using them to generate English text particularly among English language learners even for short textual interactions (like a brief regrade request)"*). However, not all instructors viewed this use positively, with one commenting (i.e. *"Students use it when they are not comfortable with their English skills, and the results of this is really frustrating/insulting to read"*).

> **Potential for Academic Misconduct:** Where instructors have observed students using GenAI tools, there are concerning reports of academic misconduct, including generating complete solutions for assignments. On the other hand, some instructors observed students using GenAI productively for debugging, generating test data, understanding code, and improving English skills.

Finally, it is worth noting that not all of the responses to this question appear to be derived from direct observation. At least one response indicated that they had no proof but were "pretty sure" (relating to academic misconduct). While instructor responses to this question do reveal the potential for GenAI tools to be used to aid student learning, they also highlight a concerning trend of academic misconduct and over-reliance. This underscores the

importance of providing clear guidelines to students in how to use such technologies productively and ethically in computing courses.

3.4.3 Student use of GenAI. Similarly, we asked students to describe the ways that they currently make use of GenAI tools in computing courses for both text generation and for code generation (Questions 18 and 19 in Appendix B).

Many of the students in our survey had not used GenAI in their courses, with around 40% of participants responding in this way for both text and code generation. This proportion was similar to that of the instructors who had reported not using GenAI. Of the students who reported not using GenAI, a small portion (fewer than 5%) refused to do so for various reasons ranging from the risks around learning to it detracting from their joy of programming (e.g. *"I do not use it at all. I love programming, I love to write programs, and I would not let anyone else do it for me"* and *"I do not use GenAI tools in computing courses at all. Struggling and debugging is a valuable part of the learning process"*). Several students were equally emphatic about not using such tools in the future (i.e. *"i will never use GenAI for computing courses for code generation"*), and one student also refused to use GenAI tools on ethical grounds (i.e. *"I do not feel that the output produced by a GenAI tool can safely be called 'my own work', when GenAI tools use so many other people's work as input to produce their result"*).

> **Student Adoption of GenAI Tools:** Most students have explored GenAI tools for text or code generation. Those who do use GenAI tools most commonly apply them for paraphrasing or summarising text, for debugging errors in their own code, and slightly less often, for code generation. However, some emphatically refuse to use these tools due to concerns around risks to learning and ethical issues about originality. This framing may help when providing course policies and explaining ethical considerations in syllabi (see Appendix D).

With respect to text generation, the most common use (reported by 20% of students) was paraphrasing or summarising existing text, for example to improve their own writing (e.g. *"I use AI to summarise my own writing to see if the point I want to communicate is clear"* and *"I will write something and then put it into ChatGPT to make it read better"*) or to produce a summary of a large quantity of text (e.g. *"I use it to write summaries about books I've been reading"*). A smaller proportion of students, fewer than 15%, reported using GenAI tools for writing new text, with responses ranging from very short descriptions (i.e. *"writing reports"*) to much more detailed processes including iterative development of written reports through multiple rounds of prompting (i.e. *"I keep sending prompts for it to change this and that, add some topics, reword some sentences, explain to me what this sentence means so I understand"*).

The most common use of GenAI by students for coding-related tasks was debugging errors in code they had written, and this was reported by 25% of respondents (i.e. *"try to fix code when not working"*, *"Helping search for bugs"* and *"I copy and paste the codes that gives weird error. I tell them what i am expecting but i am getting this then they tell me which part of codes are wrong"*). Code generation was the next most popular theme, with some students reporting using GenAI to generate solutions directly (i.e. *"If I was solving a question that I don't have the answers to, I would ask it to give me the solution"*), although others were more cautious about the outputs that were generated, with two students describing its use as a 'last resort' (i.e. *"I consider using it as a last resort. If I'm running short on ways that would solve a problem and have exhausted all the possible ideas I have then I ask for the explanation of the problem first and if that was unhelpful then I ask for a piece of code which I check for mistakes and incorporate in what I already have written."*

3.4.4 Student perceptions of the effects of GenAI tools on employment. We asked students to describe the effects they think GenAI tools will have on their prospects for future employment (Question 20 in Appendix B). There was a mix of positive and negative responses, which is consistent with the quantitative results of the corresponding Likert question (Question 17 in Appendix B).

A considerable number of students (33) seemed concerned that GenAI tools will reduce job opportunities. Several were very pessimistic and went as far as to say that all jobs will be replaced by AI (e.g. *"I think that if left unchecked as it is going right now, it will eventually take over all the jobs, no matter who you are really"*). Others were concerned that entry-level jobs will be affected more than senior-level jobs (e.g. *"competition for entry level jobs is going to skyrocket"* and *"entry-level opportunities will probably become quite rare"*). Along these lines, 19 students mentioned that GenAI tools will raise the expectations of employers and increase the difficulty of bootstrapping in the industry (e.g. *"I do believe that the standards that companies require will be higher, as AI has proved to be above mediocre, perhaps affecting juniors and paid interns"*).

Some students argued that software engineering jobs are particularly at risk. For example, a student said: *"programmers will be among the first group to see massive job losses. I believe this because the entirely text based nature of coding is well suited to LLMs. The tech industry is also faster to adapt than other industries."* Another student said: *"I'm genuinely concerned about companies realising they only need 1/5 or 1/10 as many software engineers... especially since GenAI can read an entire codebase and easily put together working code from a prompt from a senior dev that also passes tests created by senior devs"*. Interestingly, a student argued that competition for software engineering jobs will increase because GenAI will make learning programming easier, which *"will likely draw the attention of a lot of new people who would otherwise be uninterested in programming"*.

A different concern raised by some students relates to the hiring process itself. Two students indicated that the use of AI tools to *"judge résumés"* might have a *"massive impact"* on employment. According to one student, it feels *"unethical and unfair"* and it is *"incredibly draining to know that all that's between you and a job is a machine"*. Three students also mentioned that standing out to employers will become harder, given that many applicants might use GenAI tools to build portfolios that make them appear as solid candidates despite lacking the required skills. According to one student, such candidates *"will flood the job market"*, and it will be harder for future employers to distinguish between them and those who genuinely have the required skills. These concerns were recently echoed by Armstrong et al. [27] who explored the impacts of automated hiring systems as "black boxes".

While many students raised concerns, a good number of students (28) indicated that they are not concerned about AI taking over their

jobs or about the job market being significantly impacted. Some of these students questioned the ability of GenAI to perform the tasks that humans are good at (e.g. *"I do not think coding will become obsolete though, AI isn't even close to that good yet."*). Other students questioned whether the job market will change at a fast enough pace to pose a threat to their employability in the near future (e.g. *"I strongly believe that a proper programmer will most likely not be affected in terms of employment in the coming 7-12 years by GenAI"* and *"at least for the next ten years employment will be fine, my skills are transferable and I am always happy to learn new things."*).

> **Implications for Future Employment:** Students expressed mixed views on how GenAI tools might affect their future career prospects. While some believed job opportunities would decrease, others were optimistic that these tools would improve their productivity and give rise to new careers.

Beyond this, many students thought GenAI would have a positive impact on their future employment. Eleven students mentioned that they expect more job opportunities to emerge because of advances in GenAI tools, and 32 students indicated that GenAI will have a positive effect on their productivity in the workplace. In fact, expecting an increase in work efficiency was the most commonly occurring comment amongst student responses. As one student put it: *"it will make work so much easier; no more boilerplate code, or searching forever through StackOverflow"*. According to another student, GenAI tools will *"allow for more creative flow, more interesting products because the hard work can be done easier, less research time on how to complete a job, and more time completing it"*. Additionally, several students indicated that GenAI tools help them learn better, and thus will improve the skills they need to get employed (e.g. *"It can teach LeetCode pretty good :) so I'll have better chance to pass the technical interview"*). This attitude is orthogonal to that of some of the students whose responses showed negativity towards using GenAI tools while learning. According to those students, relying on GenAI tools may reduce their understanding of the material and thus affect their future chances of employment.

3.4.5 Student and instructor perspectives on when GenAI tools should be allowed. We asked instructors to elaborate on when they believe GenAI tools should be allowed or disallowed (Question 12 in Appendix C). The prevailing sentiment in the responses was that GenAI tools should not be used when students are learning the basics. Hence, many instructors indicated that GenAI tools should be disallowed in lower-level courses but allowed in upper-level courses. Some instructors argued that it is a function of complexity rather than course-level. For example, more complex assessments (regardless of the course level) are more appropriate for the use of GenAI tools than simpler ones that can be easily completed in their entirety by the tools.

Another recurring and related theme was that allowing the use of GenAI tools depends on the course and assessment learning outcomes. For example (as described by an instructor), *"if the assignment is to create a website with the goal of learning to apply HCI principles in the design, they should be able to use GenAI or other tools for the mechanical code generation"*. However, if the goal of the assignment is to see if they can write a piece of code, then they should not use GenAI tools to generate that piece of code. An instructor argued that *"this is analogous to many other practices within the University; for example, a student would not normally have to build their own computer, but if they were on a hardware design course they might have to, and submitting a purchased machine would not satisfy the learning outcomes of the module"*.

A minority of the instructor responses supported unconditionally allowing the use of GenAI tools. Some said that their use is fine as long as the student acknowledges that appropriately. Others argued that it is useless to attempt to disallow their use outside closed-exam conditions, as students will use them anyway. At the other end of the spectrum, a few responses supported always disallowing them, or disallowing them in all graded assessments (regardless of the level, type, topic, etc.).

> **Conditional Acceptance of GenAI Tools:** Both instructors and students suggested that whether the use of GenAI tools is permissible should depend on factors such as the course level, assessment type, purpose of the task, and how the tools are used. This indicates the need for nuanced guidelines and policies when dealing with GenAI in academic settings.

We asked students the same question (Question 13 in Appendix B). Student answers were more diverse and more polarised than those of the instructors. Interestingly, a good number of students (n=29) argued for disallowing the use of GenAI tools in all coursework and exams. Some also argued for completely disallowing them even outside assessments (i.e. while learning). The arguments used by these students included a range of reasons, like ethical concerns regarding how the models were trained, concerns regarding the correctness of the tools, and concerns regarding the fairness of assessments if these tools are used. However, the majority of these students argued that the use of GenAI tools *"harms learning"* and *"defeats the purpose"* of assessments. Some of these students made strong statements indicating that GenAI tools *"have no place in learning"*, are *"completely counter-intuitive to going to University"*, and are *"only used by people who aren't smart enough to solve problems on their own!"*.

Many of the students opposing the use of GenAI tools emphasised the importance of doing the assigned work and going through the full *"discovery process"* without *"taking shortcuts"*. A student said: *"effort is the road to success and minimising effort can create a generation of couch warriors"*. This comment captures the gist of many of the responses that linked using GenAI tools with deficient learning. Another recurring argument was that using GenAI tools in coursework and exams defeats the purpose of assessments. A student likened it to continuously *"looking to the back of a textbook for the answer"* and another likened it to having someone *"sitting next to you and helping you"* complete the work on which you are being assessed.

On the other hand, fewer students argued for always allowing the use of GenAI tools. An argument made by several of these students was that GenAI tools are *"the future of where the industry is going"* and thus learning how to use them is important for their success. One student said: *"when we go into employment, we will need to use whatever resources we have available to us to be as productive and efficient as possible"*.

The majority of students argued for a situational or a conditional use of GenAI tools. They provided a wide range of factors that affect (in their point of view) when the use of GenAI tools is acceptable. These factors include:

- *Course level*: GenAI tools should be allowed in upper-level courses, but not in lower-level courses when students are learning the basics.
- *Assessment type*: GenAI tools should be allowed in coursework, but not in exams.
- *Assessment weight*: GenAI tools should be allowed in minor assessments that carry a small weight, but not in major assessments.
- *Task goal*: GenAI tools should be allowed if the goal is the application of already-learned concepts (e.g. to build an artefact). It should be disallowed if the goal is learning the concepts.
- *Task size*: GenAI tools should be allowed if the task is large and complex, requiring stitching many pieces together. It should be disallowed if the task is small or trivial.

While these factors relate to the assigned task, some students felt the acceptability of the use of GenAI tools should be conditional on the way the tools are used, rather than on the task itself. For example:

- *How*: Using GenAI tools with understanding is fine. Blind copying and pasting of answers is wrong.
- *How much*: Using bits and pieces or partial solutions generated by GenAI tools is fine. Using a complete solution generated fully by a GenAI tool is not fine.
- *Why*: Using GenAI tools as a last resort, when stuck, or when there is no other way of getting help is fine. Relying on GenAI tools right from the beginning is not fine.

The last category above is interesting as it assumes that, in general, the use of GenAI tools is unethical unless it is out of necessity. The following are several quotes from the student responses that support this idea:

- *GenAI can be used as a last resort when the lecturer is rather difficult to explain a material and students use GenAI when they cannot understand what is being explained or assigned at all.*
- *GenAI should be allowed if the courses force us to do work manually without any mentoring. Vice versa, if the mentor is giving course completely i think GenAI should be disallowed.*
- *... when you run into a dead end and even after looking online and asking a friend and either don't know or you still don't understand to go and ask GenAI for an answer.*
- *I believe GenAI should be allowed sometimes when you have no one else left to ask.*

4 CURRICULUM AND ASSESSMENT

In the past 50+ years, a great body of research within the SIGCSE community addressed many trends, opportunities and challenges in Introductory Programming (CS1) courses [133]. Among these are teaching and learning approaches, new forms of assessment, shifts in content, tools, and overall course design [38]. For example, around the turn of the millennium computing educators passionately debated whether to use an objects-first approach (or not) [60]. Similarly, Alice, Scratch, Blockly, and other block-based programming languages have been the subject of much research [147, 168, 179]. Although these developments were important for many reasons and groups of (present and future) students, they are not comparable, at least not in pace and ubiquity, to the rapid changes Large Language Models (LLMs) are currently triggering in higher education, the computing disciplines, and CS1 in particular. With LLMs available on nearly everyone's phone and laptop[4], it is not only knowledge that is instantly retrievable but also problem explanations and solutions - in the form of programming code that is not necessarily correct. Given the pervasiveness of LLMs, this paradigm shift regarding the availability of knowledge, solutions, examples, and content (particularly in the form of code) is more comparable with the advent of the internet than other developments in the annals of how we teach and learn computing—yet the speed of internet adoption was much slower as a whole.

Given the myriad impacts of LLMs, it is important to acknowledge that educational systems are notoriously slow to change. Reasons for this are many, yet Lee Shulman [182] adds the pedagogical psychologist perspective, pointing out that the "signature" of a profession's teaching and learning is pervasive and perpetuated at three levels: surface, deep, and implicit (i.e., curricula, pedagogy, attitudes & values). Now, however, with the seeming ubiquity of LLMs, it is inevitable that educators consider their impact on teaching, learning, assessment, and delivery, leading to possible redesign of their courses at all of these levels. Additionally, perceptions and experiences of both students and educators also vary, often focusing on difficulty [35, 132] and LLMs may stand to impact this long-standing issue.

Based on the concept of Constructive Alignment [42], learning objectives need to be aligned with exercises, assignments, and assessment methods. Therefore, we discuss the relevance of LLMs for CS curricula and assessments with regard to course objectives, and course activities including formative and summative assessments. This discussion is centred on expert interviews we conducted with introductory programming educators, focusing on their changed educational views and practices to highlight how computing education is evolving (with what seems to be lightning speed) in light of LLMs.

4.1 Methodology for expert interviews

To understand how computing curricula and assessments are currently being affected by the emergence of LLMs, we conducted an interview study with computing educators as experts in the field. The interviews were semi-structured, with an interview guide as a basis. Using the purposeful sampling method [146] led to the selection of experts via the authors' networks, who were contacted via email. Moreover, an invitation was sent out to active contributors to a discussion thread from the SIGCSE mailing list concerned with LLMs. Another recruitment attempt was made via an open question

[4]Acknowledging that internet access is required to access LLMs, and that subscription-based services which could be superior to free ones present issues of access based on means, and potentially opening new divides.

in the instructor survey, where respondents willing to elaborate on their responses in an interview could enter their contact details (see Appendix C). The most important criterion for inclusion was that educators would have concrete plans or views toward changing their current course structure, assessment, or classroom practices in light of LLMs. This is one of the main ways the present work differs from that of Lau and Guo [121] discussed in Section 3.1.

The interview guide included the following questions and follow-up questions:

- Which course(s) are you teaching in the next semester? [If they are teaching multiple courses, then try to talk about one particular course or at least make sure it is clear which course is under discussion at any point in the conversation.]
- Do you have an explicit set of written learning objectives/ competency goals for this course?
  - If yes, will LLMs change these goals?
  - If yes, what goal will change or be removed? What goal(s) will you add?
  - If no but they have informal learning objectives, ask how they think these will change (or have changed).
- Are you planning to change your pedagogy and/or learning activities because of LLMs?
  - If yes, how (what did you use before, how do you change it, and why exactly)?
- Are you planning to change the assessment mechanism?
  - If yes, how (what did you use before, how do you change it, and why exactly)?
- What is your vision for that course in the context of LLMs?
- Which opportunities for enhancing teaching, learning, and assessment can you think of?
- Which challenges come to your mind if you think about LLMs in the context of computing education?

These questions resemble some of the survey questions, but they allow for a more in-depth elaboration of instructors' practices. Interviews were scheduled to last between 20 and 60 minutes. In practice most were closer to 60. They were conducted via Zoom and automatically transcribed via speech recognition software, followed by a correction loop by a human (interviewers checked transcripts of other reviewers, not their own). After the transcripts were finalised, we deleted all audio and video recordings in accordance with the protocol submitted to the University of Toronto Research Ethics Board, who approved it prior to the study. Respondents were free to decide if they wish to remain anonymous or to be named. Those that are named in this report gave their consent for this. Affiliations are noted in the acknowledgements section and with the interviewees' first mention in this section. We also allowed participants to review a draft manuscript before final publication, to ensure that they are not misrepresented.

The sample comprises 22 computing educators from nine countries and five continents. Table 6 shows the locations of the interviewees.

The interviews, which reflect practices from the teaching community in this new context of Generative AI were fully transcribed verbatim and served as a basis for thematic analysis [47, 170]. Initial themes were deductive, rooted in the literature review (see Section 2). As a next step, themes were reviewed and refined inductively based on the interview material, resulting in the structure presented in the following sections. Two of the authors reviewed both the verbatim interview transcripts and the identified themes for all interviews. Next, themes were discussed and specified among four of the authors. Interview summaries were then compiled based on these themes and double-checked against the themes and transcripts by a second author.

Table 6: Countries of Interviewees.

Country	Count
Austria	1
Brazil	1
Denmark	1
Germany	2
India	1
New Zealand	1
The Netherlands	1
UK	2
USA	12
Total	22

4.2 Learning objectives

The rise of LLMs is causing many instructors to rethink their course learning objectives. For instance, as students increasingly use LLMs to write programs, instructors might need to put less emphasis on writing code and more emphasis on reading code. Some researchers have begun to investigate this new emphasis [33, 188]. Although the ability to both read and trace code has always been important [19, 128], an examination of course syllabi showed that this is often an afterthought [37, 112, 114]. Similarly, many topics in introductory programming and computing in general have been heavily researched up to now, yet their effects on the curriculum vary [38, 59, 111]. Large language models have been recently linked with many of these topics in various ways [33, 191]. Such topics include metacognition [73, 130, 156], algorithmic/computational thinking [31, 171], communication skills [154, 193], and their respective dispositions.

In 2019, Becker and Fitzpatrick [37] collected syllabi from 234 introductory programming courses from around the world and made their data available publicly for others to use (and contribute to). One of the types of data they categorised was explicit learning outcomes. Of the 234 syllabi, 154 contained explicit learning outcomes. The five most common learning outcomes were: "testing and debugging", "writing programs", "selection statements (if/else, etc.)", "problem solving (including computational thinking terms)", and "arrays, lists, vectors, etc.". These five objectives appeared on at least 40% of the syllabi. Looking through the full list of learning objectives, the only one that was directly related to reading code was "tracing program execution", appearing on only 3% of syllabi. Kiesler conducted a similar study in 2022 [112, 113] using syllabi from 35 German universities. She found that the most common objective was "writing code" and that the objective "being able to

read, explain and identify the output of (foreign) code" appeared on less than 10% of syllabi.

In response to the rapid advance in LLM capabilities, educators are reconsidering their courses' objectives. In the following subsections, we present the respective themes identified in the interview transcripts and relate them to some recent studies in the field.

4.2.1 How instructors are changing their learning objectives. Several instructors discussed changes in their upcoming course learning objectives. However, given the rapid emergence of LLMs in computing education, in most cases, these changes are not yet reflected in official curricula or course syllabi. Instead, some educators are changing more fine-grained learning objectives in their courses.

James Davenport (University of Bath, UK), for example, introduced two new sessions concerning the impact of LLMs on cybersecurity in his class. Even though the official course objectives remain as they were due to their general nature, Davenport has started to teach students how defenders and attackers could take advantage of LLMs.

Many educators acknowledge the dynamic nature of technology and anticipate potential adjustments to their learning objectives in the near future. Viraj Kumar (Indian Institute of Science, Bengaluru, India) expressed the need for flexibility, recognising that changes might be necessary as technology evolves even further: "*And even now I'm sort of holding my breath because now I'm saying, hey, let's put out these things, but you know, maybe things change.*" Statements like this reflect the fast pace of advancing technologies in computing education and educators' openness to adapting their practices. Kumar recently updated their CS1 class of approximately 50 students to include the topic of LLMs' role in code generation.

Educators like Ewan Tempero (University of Auckland, New Zealand) emphasise the role of LLMs in automating routine tasks, enabling educators to shift their focus toward nurturing critical thinking skills: "*The more tools that [students] have to support doing the stuff that really isn't that interesting, the more [educators] can focus on the interesting stuff like critical thinking.*" In this context, Briana Morrison (University of Virginia, USA) highlighted the importance of using citations with LLM-generated code and teaching students to evaluate LLM output in a critical manner. Even though educators are not teaching students how to write prompts at the University of Virginia, they are "*going to have a statement in the syllabus that using LLMs is allowed. However, we are going to require a reference, like a citation, that says this was generated by an LLM.*"

Some educators, including Leo Porter (University of California San Diego, USA) note the importance of prompt engineering when using LLMs. Porter introduced new learning goals for his CS1 class, emphasising the non-deterministic aspect of LLMs, prompt engineering, and problem decomposition. Porter feels that crafting effective prompts is a competency students should develop as students should learn how to interact effectively with LLMs, ensuring that they obtain meaningful and accurate results. As for problem decomposition, Porter said: "*We didn't used to teach problem decomposition"* but felt that this was part of their hidden curriculum that he and his colleagues are pleased to see moving to the forefront.

In this context, Michael Kölling (King's College London, UK) adds that LLMs might force us to look at learning outcomes at the program level instead of the course level and that using LLMs should be explicitly taught. This aspect is also emphasised by Kristin Stephens-Martinez (Duke University, USA) who said "*We're going to have to help students understand how to use LLMs ... and you all [the students] need to understand that ChatGPT is fallible, and you need to be very critical of what it's doing.*" Rodrigo Duran (Federal Institute of Mato Grosso do Sul Brazil, Brazil) explicitly encourages students to test and understand LLM-generated code, enabling them to evaluate if the LLM answers are correct and to adapt their answers accordingly.

A recurring theme among the educators we interviewed was the elevation of code comprehension and critical thinking skills. Jean Mehta (Saint Xavier University, USA) highlighted the need to assess students' ability to read and understand code more thoroughly, a competency that has often been overlooked in the past. Mehta concludes that "*we should have more time to spend on these kinds of things.*"

It is possible that the impact of LLMs on introductory programming might go beyond changing course objectives and lead to the introduction of entirely new courses, syllabi, and learning objectives. An example of such a recent development is the "Generative AI" online course by DeepLearning.AI [65].

4.2.2 Preserving core learning outcomes. Computing education has always experienced change over time as new tools and technologies were introduced. Although LLM-based tools like Copilot may serve as a useful springboard for solving CS1 problems, students still need to dedicate time to learning algorithmic thinking, program comprehension, debugging, and communication skills [191] in order to become not only proficient computing experts but also to use LLMs effectively. A number of interviewees shared this perspective and highlighted the need to preserve several core learning outcomes.

Educators who do not alter their learning objectives stress their focus on teaching fundamental programming concepts. The learning objectives of Peter Mawhorter's (Wellesley College, USA) introductory CS1 class remain stable and focus on fundamental concepts. Similarly, Frank Vahid (University of California, Riverside, USA) believes that it is still important to teach students how to define variables, use branches or loops to solve problems, how to use functions to keep code modular or use C++ vectors to store data. Thus, students still need to learn to code and practice. In the era of LLMs, Vahid thinks "*... the most pressing thing is making it [assessment] harder... it [Generative AI] takes the work out of homework.*". Vahid pointed out that students have for some time been getting help through other means such as Discord forums, Chegg, Stack Overflow, Piazza, etc., noting that "*We've had a leaky roof for a long time. And LLMs are the storm that finally causes us to be flooded.*"

Similarly, Leo Porter, who recently published a textbook with Daniel Zingaro for teaching introductory programming with the help of LLMs from day one [155] emphasised the importance of preserving core learning objectives that focus on teaching fundamental programming concepts. Porter noted that these objectives remain unchanged due to the foundational nature of the skills taught. They conclude that "*... things are staying the same for the Intro class because the skills I'm teaching there are so basic.*"

Some educators, including Mark Liffiton (Illinois Wesleyan University, USA), view LLMs as valuable tools that can aid students in

coding tasks. Liffiton intends to maintain hands-on coding practices while integrating LLMs into Programming Languages and CS2 courses. They focus on building foundational knowledge that complements the capabilities of LLMs: "*I still want the students to be doing that work. I still want them to be practising the things that the tool could do to build the base of knowledge so they can later do the things that the tool can't do.*" Another interviewee shares this concern: "*I'm very worried that if everybody forgets how to write code and think through this stuff, we will lose the ability to make new things.*" At the same time, educators seem to be aware of the need to prepare students for industry expectations, which is likely to include LLM use.

Dan Garcia (UC Berkeley, USA) pursues a similar approach. He recognises the potential of LLMs as educational tools, or aids, but emphasises the importance of teaching programming basics first. Garcia encourages the use of LLMs once students have mastered traditional programming concepts, concluding that "*... we can't stop teaching kids how to program.*" This perspective is shared by Austin Cory Bart (University of Delaware, USA) who is not changing the learning goals in an introductory programming class for about 280 students. Nonetheless, they expect adjustments in subsequent courses as LLMs become more integrated into programming practices: "*But I look at almost every single course after mine as, oh, yeah, that's probably going to need to change learning objectives.*"

To conclude, computing educators express the need to balance between leveraging LLMs for problem-solving while ensuring that students continue to develop competencies in algorithmic thinking, program comprehension, and debugging. Educators vary in their approaches, with some maintaining a focus on teaching fundamental programming concepts, while others see LLMs as complementary tools to enhance learning. The preservation of core learning objectives, particularly those related to basic programming, remains a consistent concern among educators.

4.2.3 Towards conversational computing. If computational thinking is the learning goal of a non-majors course, then using an LLM-based tool such as GitHub Copilot may be a useful approach, as advocated by Denny et al. [70]. For students in non-computing majors who currently only take one (or just a few) programming course(s) to learn enough to write simple programs, it may be that using an LLM tool is all that is needed, making a programming-specific course unnecessary [152]. It may even be more effective than typical courses at introducing these students to programming and may also broaden participation in CS courses in the future.

Michael Caspersen (It-vest & Aarhus University, Denmark) believes that LLMs are forcing us to rethink what we actually teach our students, and encourages us to view them as an opportunity. According to Caspersen, LLMs do not add something qualitatively new, but quantitatively indeed! They emphasise issues that have always been present. Hence, LLMs may even contribute to increasing the quality of computing education, thereby making it more attractive to a broader range of students.

4.3 Course activities

LLMs can be useful in generating several kinds of learning activities, including novel variations of programming assignments [174]. However, this may introduce problems such as ensuring that students have been provided appropriate content in order to understand novel variations. LLMs can also generate good explanations of code [94, 123, 137, 174]. This provides a mechanism for novices struggling to understand the run-time behaviour of novel problems to get auto-generated and hopefully helpful explanations. This use-case exemplifies the potential for LLMs to ease the burden often felt by teaching assistants, and could be a first line of help for students [174]. Another type of learning activity that can be provided by LLMs occurs in settings where students use them to help generate code solutions. Here, students can use LLMs in an iterative improvement loop. Students can continually alter prompts to refine the model output helping them to 'build-up' a solution [196]. In their interview Mark Liffiton took this concept one step further stating *"So they're great educational tools because they can give something akin to one-on-one tutoring... and I think there's a ton of value in there."* This particular use-case of LLMs is less about what LLMs can produce, but what they can do for students (and educators) and aligns with what the Artificial Intelligence in Education (AIEd) community has been discussing for years [34, 96, 97, 131].

Liffiton has also developed a tool called CodeHelp [127] that uses LLMs, which he is going to use with his students. The tool can do some of what ChatGPT can do, but specifically does not solve the problem, and does not give students complete solutions - a kind of "sanctioned" access to the educational power of LLMs. This could combat the concern of students becoming over reliant on them, and not actually learning from them. Mark sees this as adding to the course learning objectives to include working with this new tool and therefore LLMs. Similarly, Frank Vahid sees LLMs as tutors that will be available around the clock, and importantly, as tutors that will not judge students, stating *"I'm very hopeful that it will become another TA for the class."*

An important aspect of teaching is using carefully crafted examples to illustrate salient points. If the goal is to use a *real* or *running* example, it can be tedious to have to deal with the many irrelevant (to the example at hand) aspects of a problem in order to ensure the example compiles and runs, in addition to keeping focus on the point desired. LLMs can help instructors with the tedium of such minutiae [137, 181].

Many of the student-initiated possibilities discussed above come with academic integrity concerns we discuss in Section 6. Several interviewees were aware of this, stressing the need for educators to emphasise the importance of students doing their work.

Perhaps the most extreme example of how Generative AI might be used in the introductory programming course comes by way of Daniel Zingaro and Leo Porter's new textbook [155]. The book begins by introducing students to the GitHub Copilot plugin within the Visual Studio Code IDE before students have learned to write a single line of Python code. Students create their first programs by typing English comments and letting Copilot generate the code. This corresponds to the *sketch model* from Alves and Cipriano's Centaur Programmer [24] where the programmer generates the outline of the solution and the AI fills in the gaps. As Zingaro and Porter explain each line of LLM-generated code, they use the opportunity to teach the related Python syntax and programming concepts. But

before they do this, they introduce functions and use this to motivate top-down design. Porter will be teaching 700 students using this approach in the upcoming semester (September 2023).

4.3.1 In-class Activities. LLM-based tools have motivated some changes to activities that specifically happen during classes. Denny et al. [70] found that Copilot's performance is substantially improved when it is prompted with individual problem-solving steps in natural language and they explicitly encourage teaching prompt engineering to students. David H. Smith IV (University of Illinois, Urbana-Champaign, USA) will be using a tool and specific exercises for students to practice LLM prompts so they can use LLMs effectively. Leo Porter also stated that they will be adding a learning goal on prompt engineering as discussed earlier.

Although not planning on getting into prompt engineering, Briana Morrison is going to do more live coding with LLMs in the classroom including analysing why certain code is wrong. Viraj Kumar has also used Generative AI for live coding. Jérémie Lumbroso (University of Pennsylvania, USA) created guided sessions using LLMs and shows examples of them giving incorrect answers. Similarly, Kristin Stephens-Martinez told us that they plan to work out examples in advance of using prompts that demonstrate a hallucinated answer or other technically incorrect responses to help students see for themselves that LLMs are not oracles. Austin Cory Bart is going to demonstrate the use of LLMs in class for three reasons: First, because LLMs are a great way to introduce AI topics, providing a way to bring advanced material into the introductory course so students can look forward to what is coming later. Second, Bart feels that if it is not discussed with students they will just use it anyway, but in a more misguided manner. Finally, Bart stated: *"At some point we have to start incorporating these tools... programmers are going to be putting these tools into the workflow. And I see that on my own... when I'm coding this summer having co-pilot auto-complete an entire function that I just started writing, that's too much of a game changer for it not to be addressed."*

Interestingly, Leo Porter is the only interviewee who mentioned pair programming directly. Porter plans on continuing to use pair programming and peer instruction in class. Dan Garcia noted that LLMs should be used as nudgers, hint-givers, or help-givers, but not oracles—seeming to match the desired role of a human pair programmer. Michael Caspersen also noted that LLMs should be integrated into peer-to-peer learning. Other examples in the literature have speculated on what may come of pair programming in light of LLMs. For instance, Dakhel et al. noted that when used by experts, Copilot can be an asset as its suggestions could be "comparable to humans" in terms of quality. However, it could become a liability when used by novices who may fail to filter its buggy or non-optimal solutions due to their lack of expertise [61]. Wermelinger stated it is not surprising that GitHub dubs Copilot as "your AI pair programmer", even though the interaction is far more limited than with a human—notably, Copilot does not provide a rationale for its suggestions [191]. However, Frank Vahid did mention a positive aspect of how many Generative AI tools present their output, noting positively that LLMs generally do not judge the student.

However, in the same way that they expect students to learn to use an IDE on their own time, some instructors have indicated that they will not be dedicating classroom time to explicitly teaching students LLMs. Rodrigo Duran is not actively encouraging LLM use but at the same time is not actively discouraging their use. Dan Garcia does not plan on incorporating LLMs into their course, at least for the time being, choosing to keep their "ear to the ground" and see what others are doing, and stressing that the fundamentals are important: *"Do we stop teaching long division now that we have calculators? Do we let students use calculators when they are learning long division? No and no."* Peter Mawhorter does not plan to allow LLM use in their course out of fear of harming particular students, expecting that a small percentage of students—likely those who are marginalised in other ways—will have bad experiences due to biased, possibly sexist, or racist output. It is also possible that these tools will give different outputs to different students, for instance, based on the student's name if provided in a starter code or a prompt.

4.3.2 Unsupervised Activities. Student use of LLMs in unsupervised learning activities, self-assessments, and other, self-directed scenarios have also been the subject of research. MacNeil et al. [137], for example, used GPT-3 and Codex to generate three types of code explanations (line-by-line explanations, lists of important concepts, and high-level summaries) from code snippets from an instructor-developed web software development e-book. They found that Codex generated less helpful and more verbose explanations than GPT-3. Moreover, Codex included code in the explanations, even though it was not desired. Students rated these explanations on a 5-point Likert scale, confirming that explanations matched the code and were useful for learning in general, whereas line-by-line explanations were rated as least helpful.

An exploration of the potential of LLMs to generate formative programming feedback [115] suggests that ChatGPT performs reasonably well in generating feedback to students' solutions to introductory programming tasks, indicating that students can potentially benefit in unsupervised learning scenarios. However, it may fall on educators to guide how to use the generated feedback, as it can contain misleading information for novices. Several of the approaches discussed in the prior section are aimed at mitigating this.

Interviewees mentioned unsupervised activities less than in-class activities in general. This could be at least partially due to the fact that this (Fall 2023) is the first academic year where Generative AI could be considered mainstream and unsupervised activities are more difficult to plan and hypothesise about. Most of the discussion around unsupervised activities is centred around ensuring that students are doing their own work (even if using Generative AI as a tool for help) and that students understand the code that these tools produce. Frank Vahid believes that this could lead to increased instructor emphasis on tools that analyse student behaviour.

In a high school introductory programming course, Christian Tomaschitz (Theresianum Eisenstadt, Austria) had half of his students use ChatGPT and the other half use an e-book. ChatGPT was not introduced to students and students registered OpenAI accounts with their school e-mail addresses. All students had the same exam which was half open-book (including internet access), half closed-book. Tomaschitz noticed that the ChatGPT students had the possibility to interact (with ChatGPT), and these students were faster solving the problems, but it seemed as if they did not

critically reflect on the output, and understood less. Tomaschitz was concerned that the ChatGPT group was over-reliant on the tool and did not understand the output as well as the e-book group.

Viraj Kumar used GitHub Copilot during class for live coding and told students they were free to use it, or any other Generative AI for coding. Students were polled after the course and asked if they were using ChatGPT. Most said that they were not. The consensus explanation seemed to be that they wanted to learn programming themselves and their thinking seemed to be that when they go for technical job interviews they will be expected to code without assistance.

Several interviewees including Leo Porter and Michael Caspersen believe that LLMs will cause educators to focus more on code reading and less on writing from scratch. Caspersen noted that students read more than they write when learning to read natural language, but to-date this has not been the traditional approach when it comes to programming. Further, LLMs could support a "use, modify, create" approach for programming. Briana Morrison envisions more exercises about why certain code is wrong and fewer on writing code from scratch.

Kristin Stephens-Martinez mentioned that they are concerned about LLM use by novices pushing more metacognitive practices earlier in the curriculum, perhaps before students are ready for this. For instance, students will have to ask themselves "am I going to take this shortcut or not?". Getting students to recognise when it is a shortcut versus when it actually is not a shortcut is something that novices typically are not in a position to assess.

4.4 Assessment

Although we previously discussed activities that could have been assessed, in this section we focus specifically on formal assessment. Research on LLMs has demonstrated their ability to answer typical CS1 assessments that involve writing simple functions [70, 117] as well as more advanced material. Similarly, it has been shown that they perform in the upper quartile of real students on CS1 exams [85]. It has also been shown that they perform just as well on CS2 exams as CS1 exams [86] suggesting that LLMs may soon be capable of effectively solving even more advanced problems. Indeed, recent results with GPT-4 suggest that it can solve most exercises in introductory programming courses [175], which is also supported by our benchmarking work presented in Section 7.

However, evidence also shows that LLMs do not perform as effectively on computational thinking problems that do not involve code writing [39]. Similarly, while LLMs can often correctly answer more than half of coding-based multiple-choice problems, they answer double-digit percentages of multiple-choice problems incorrectly, leading to the hypothesis that either a combination of natural language and a code snippet, and/or chain-of-reasoning steps pose a challenge for LLMs [177]. It might be tempting to think that even though LLMs can do well on relatively simple well-specified functions, longer and more complex problems are beyond the capabilities of LLMs—although LLMs have been shown to perform well on more advanced (coding competition) problems [126].

4.4.1 Exams. Some instructors are moving towards invigilated exams and assessments, and these may be worth more marks. For example, one interviewee changed the grade weighting of their programming assignments from 50% of the course grade to 0% of the course grade. They added an assessment category, "coding interviews", which ensures that students are not using LLM tools as part of the assessment.

Several educators mentioned oral exams. Jean Mehta plans to use 20–30 minute one-on-one oral exams at the end of every section. This is made possible by a combination of a flipped classroom with many videos and book/autograder technology such that there is less need for traditional lectures covering the material. Michael Kölling said, "*I think there needs to be at least some assessment that includes an oral element because you know, there is at the moment, the problem is that submission of written work, whether it's text or programs, is taken as a proxy for intellectual achievement, right? And what we actually want to assess is intellectual achievement. We want to see some, you know, sort of intellectual work having happened there and we take the written work as evidence of that. And you know, these tools have removed that connection... The creation of written work is no longer evidence of intellectual work having happened.*"

Similar to personalising assignments, on open-ended writing assignments, Ewan Tempero requires more specific answers directly related to course materials: "*And it's not that we used strange terms or unusual terms, but we used specific terms in a particular way. And so, we expected answers to reflect that, to demonstrate that, yeah, they did actually understand what the course material was.*"

4.4.2 Homework. There may be less summative unsupervised work because instructors no longer trust that unsupervised work is the students' own. This may be accompanied by a tendency to not grade code assignments going forward. Along with increasing the weight of exams, many teachers are devaluing "homework" assignments altogether. Mark Liffiton said, "*I will sort of be operating in this assumption that some students may end up just getting a tool to do the work. And thus, I don't want to be putting too many points on that and giving an unfair advantage in those cases.*" One instructor decreased the grade weighting of their programming assignments significantly and added this text to the assessment description, "*As the programming assignments are intended primarily for practice and learning, your program does not have to be fully correct to receive credit. The final evaluation of whether you have learned from your programming assignments is in your ability to solve problems on quizzes and the exam.*"

Some teachers no longer require "writing assignments" in the traditional sense. Jan Schneider (Goethe University, Germany) has a course that used to include writing a scientific paper, but it has been changed to allow students to produce it using LLMs. "*I mean, the main objective is to help people to start thinking in a more scientific way. That's the overall objective.... I will not ask them to develop a mini paper by scratch where they need to write everything.*"

4.4.3 Process over product. Rather than assessing final solutions as products, some educators are increasingly focusing on assessing learning processes (which is common in other disciplines, e.g., teacher education) such as submit-in-stages; solution reflection, commentary, critique; interviews; portfolios or learning journals; and presentations. Taking the approach of having students focus on the learning process through a diary or journal, Christian Tomaschitz replaced several previous exercises with reflection assignments in a diary for students to document their learning process.

Related to more open-ended assignments, Leo Porter said, "*... gone are the days of us really just completely describing the exact behaviour of the functions... And then it's going to be a lot more work for us to grade because we're going to have to now look at the code or the PDFs of the code. Or look at the video of them showing how the code works.*" This approach also shifts focus from memorising knowledge (or in a rote manner, the process that leads to the product without questioning the process) to the application of skill and critical thinking. Jérémie Lumbroso applies this shift to his Discrete Mathematics course, explaining that "*the idea is to focus less on the product and to focus more on process, on making sure that the students are able to explain what they are doing.*" Some educators have already started using LLMs as part of the learning process. Briana Morrison explains a possible assignment: "*Here's the problem. Here's the prompt we gave ChatGPT, or Copilot, or whatever. Here's the output it gave us. It's wrong. Tell us why it's wrong.*"

Several educators worried that some classes do not lend themselves to the approaches mentioned above. For example, elementary theory courses are not amenable to "personalise" a proof. Michael Caspersen, on the importance of basic competencies, even in the face of powerful LLMs stated "*And that means that if you add a good programmer to large language models, you get two good programmers. Basically, that's the equation, right? If you add a mediocre programmer to large language models, you get just large language models. So, from that perspective... if you want to really be able to amplify the capabilities of humans, we need to make sure that the basic competencies remain.*"

Frank Vahid stated: "*The biggest challenge is cheating ... you've gotta learn ... you've got to work to learn. You can't just let tools do your work for you... and this is true beyond computer science. In English, you've got to learn to write. Even though ChatGPT can do most of your writing for you... you've got to learn to write... that's how you think. That's why you're valuable as a human to a company... and so the same with computer science. So somehow, we've got to get that message out. It's just so tempting for students to save so much time.*"

It is worth noting that some of these findings support those of Lau and Guo [121], notably the increased emphasis on invigilated exams, oral exams, and process-based assessments.

4.5 Institutional initiatives, policy, and context

Although some institutions (at least initially) have universally banned the use of LLM tools in student work [142], others are starting to embrace them. There is little doubt that this will lead to an array of policies and initiatives that may be at the university, faculty, or class levels. Given that public awareness about LLMs occurred mid-academic year for many institutions (almost universally for North America and Europe in addition to many other parts of the world) this September (2023) marks the first academic year where LLMs are a nearly ubiquitous topic. Given that institutions are typically slow to react mid-year—if they react at all, we have yet to arrive at a steady state in terms of institutional initiatives. Nonetheless, they are beginning to emerge as educators start thinking about the coming academic year.

Peter Mawhorter's local policy is that LLMs are not allowed in class. In delivering this message to students Mawhorter aims to discuss with students why that policy exists. On the other side of the coin, Leo Porter is embracing LLMs and is aiming to use them, for instance, in quizzes. However Porter's institutional challenge is finding and organising enough computer-based testing facilities. This is one example of how LLMs can have knock-on effects that could not only affect the course in question but others via resource allocation and timetabling. Michael Caspersen set out a middle ground, starting with basic competencies and gradually building-in the use of LLMs letting higher levels of the SOLO taxonomy [43] come into focus.

Sven Strickroth (LMU Munich, Germany) noted another challenge that is caused by institutional policy—significant changes to learning objectives are not easily possible due to the local accreditation cycle which dictates that this occurs only every five years in many institutions.

The knock-on effects of LLM use also need to be considered. For instance, Leo Porter is aiming to use a modified version of PrairieLearn[5] that supports LLMs for quizzes. However, finding and organizing enough computer-based testing facilities is (to date) a challenge as computer labs and institutional machines have become less frequent over the years.

4.6 Other challenges and opportunities

At the end of the interviews, we asked interviewees for their views on the challenges and opportunities they foresee in terms of the effects that LLMs will present in introductory programming courses. While some of these have already been discussed, several have not, and are presented here.

4.6.1 *Challenges.*

(1) Jan Schneider noted that everyone (including LLMs) have a lot of blind spots, and these can be found by writing—either in code or in natural language. Blind spots often appear when other people (or LLMs) try to understand what we wrote. This can feel like, "*whoa, there's a blind spot... something I didn't know that I was missing... and I have a big fear that if we start using these large language models, we will never acknowledge our blind spots. And we will miss a lot in learning and developing.*"
(2) Mark Liffiton mentioned that it is a challenge now to make sure students are learning things that they could just have a tool do faster. The concern of over-reliance—not learning the things they would have if they did not use the tool to do it for them—feels like cheating in a way because it is not learning the things you would if you had done the work yourself.
(3) Frank Vahid mentioned that "*you need to think.*" A human is only useful because they are clever, and thoughtful, and intelligent. That is why we teach them programming—because it is a way to help them learn to solve problems. Even though we have calculators we still should know how to do arithmetic, despite the existence of calculators.
(4) Michael Caspersen fears that LLMs will enable disciplined students to become better, but that it will be a danger for undisciplined students who are seeking the easy way out.

[5] www.prairielearn.com

Caspersen noted that it should not be considered the student's fault for taking the easy way out. It should be turned into a challenge for educators to come up with assessment systems that do not have an easy way out. Caspersen also noted that in many ways LLMs do not add something qualitatively new, but emphasise issues that we have been dealing with for a long time. This was corroborated by Michael Kölling who stated that it is the scale of these issues exploding that is novel, for instance the illusion of achievement (which is nothing new) and what that could do to learning at scale. Kölling is also concerned with intellectual laziness noting that learning is a struggle—learning only happens when you intellectually struggle with something, and if LLMs offer an easy way out, then is learning happening? Kölling also mentioned cheating as a challenge, but that this is obvious, boring, and solvable.

4.6.2 *Opportunities.*

(1) Michael Caspersen sees a big opportunity in terms of rethinking what we are doing. *"We should think deeply about what we actually teaching our students - and LLMs are doing that... [LLMs] radically challenge our reflections on what to teach, what to assess, how to assess. So that's a great opportunity."*
(2) Michael Kölling sees individualised learning in terms of progress, interests, feedback, and help as a big opportunity, noting that humans saw similar issues with books and the printing press. There was concern that people would not need to remember anything anymore. However, we lived. "*In fact, books made things better, right?"*
(3) Dan Garcia sees potential in scaling support which benefits educators, institutions and students, positing *"Imagine an LLM that could examine every exam script and where mistakes were made give a whole concept map of what went wrong in the notional machine and where and how. I can try this for a few students but I have over 1,000. This could help scale support for everyone."*

4.7 Discussion

Above we have discussed issues raised by the expert instructors that we interviewed. In addition to those, we consider a most certainly non-exhaustive set of issues we believe are important for instructors and CS program designers to consider presently.

The potential biases reinforced by models trained on large datasets [77] are concerning. This could be especially important for instructors who use LLMs to create course materials. Another issue is the presence of hidden or implicit learning objectives in existing computing programs—something Leo Porter described regarding problem decomposition in Section 4.2.1. There is little doubt that there are other such hidden learning objectives spanning not only knowledge, and skills, such as reading and tracing code but also dispositions, inter- and intra-personal competencies, and other aspects relating to the whole person [163] which the emergence of LLMs might bring to the fore. For example, many programs expect their graduates to be comfortable developing large programs using (new) debugging tools, work in a self-directed manner but also perform well in teams, and overcome challenges to pursue their goals long-term. But curricula often do not explicitly include these program-level outcomes in the learning outcomes for a particular course [38, 113]. The same is true for course activities and assessments. As individual instructors respond to the landscape changes induced by LLMs, it becomes more important than ever to consider and implement the constructive alignment [42] of individual course activities and desired program-level outcomes. This is not a new concern for educators, but one that has been amplified by the rapid change in teaching and assessment settings we now see happening (or about to happen).

Related to this is the potential for LLMs to change the workplaces into which we are graduating students. While a number of researchers have looked at how current developers may use LLM-based tools [32, 52, 81] and how LLM-tools may be incorporated into professional software development tools [84], the participants in these studies have been programmers who learned to code initially without using LLMs. Although professional developers may benefit from AI's human-quality suggestions, novice developers lack the expertise to recognise and understand buggy or non-optimal solutions [61]. The use of AI tools could become a liability if inexperienced developers fail to remove or correct the tool's incorrect suggestions [61]. Potentially more dangerous is the fact that students can now execute code that they do not understand, yet designed through natural language prompts.

It remains to be seen whether students who have LLMs available from the start will have the discipline and dispositions to develop programming competencies deeply or whether this will even matter. Reminiscent of the 1990s and early 2000s discussions of objects-first or objects-later, the opportunity to focus first on top-down design but have a working code for interesting problems completed by the AI is not universally recognised as a positive development. It is also interesting, given how little is known about the true role of notional machines in the classroom [76], that generative AI is likely to alter how students conceptualise the computer and program execution.

LLMs also impact peripheral and applied computing fields. For example, it was shown that LLMs can successfully solve 97% of the programming problems in a bioinformatics course [152]. The authors conclude that the models perform so well that bioinformatics students in the near future may no longer need to know how to write (and likely not understand) code. As a consequence, several questions arise on the effects of LLMs not only on other applied areas of computing (e.g., data science; digital forensics; security; and games development) but also on programming as one of the core tiers of every computing degree. Related to that are questions about how the introduction of LLMs might affect participation in the computing field. Perhaps reducing the focus on syntax will make the field more attractive to traditionally under-represented audiences and increase retention rates. Additionally the influence of media coverage of computing topics is known to be a large factor in the decisions pre-university students make in terms of what courses to pursue at university. It remains to be seen what effects the intense media hype surrounding LLMs will have on future computing intakes.

We anticipate more changes in learning objectives, course contents, learning activities, and assessments, which will, in turn, affect whom we teach and why in the (near) future. This might go hand in hand with long overdue changes in computing's signature pedagogy [182], and its implementation on the surface, deep, and implicit

dimensions. For decades, computing education researchers have presented excellent research on how to better teach our discipline, yet much of this has not made it into practice. We believe that the emergence of LLMs may finally force much-needed (and long ago intended but not yet fully implemented) change.

Advice for educators:

- Acknowledge the existence of LLMs with your classes regardless of whether you embrace them or do not allow their use.
- Make clear and discuss institutional and class policy, what it allows, what it does not allow, and why it is that way.
- Assume that students are using LLMs even when not permitted.
- Do not underestimate the ability of LLMs to produce solutions to your activities (which may be indistinguishable from student-generated solutions).
- Consider using an LLM tool to help generate course materials. If you do this, be aware of possible bias in the output.
- Reconsider your learning objectives in terms of their relevance to preparing those students who are aiming for careers in the software development industry (which is increasingly making use of LLMs in day-to-day work).
- Reconsider your learning objectives (e.g., reading and understanding code), learning activities, and assessments to assure your courses remain constructively aligned.
- Interrogate your learning objectives and ask what might be hidden or implicit and which LLMs might provide a vehicle for more focus. Correspondingly, interrogate your learning outcomes and ask which might be over-emphasised (e.g. code writing) that might need to be balanced with those that LLMs bring to the fore.
- Consider using LLMs in your course if only to provide a chance for students to receive more feedback, and practice independently, provided they are equipped to interpret LLM output in a way that facilitates learning.

4.8 Limitations and threats to validity

We used three sources to build a list of educators to invite for interviews, the SIGCSE mailing list (via a reply to a message about LLMs), the opportunity for those responding to the instructor survey to volunteer to interview, and authors' own networks. Perhaps as a result of this our geographic representation is skewed. As expected the United States forms the bulk (55%) of responses. Although our interview pool spanned five continents, we had no interviewees from Africa and only one interviewee each from Asia (India), Oceania (New Zealand), and South America (Brazil). Unfortunately, only 14% (3) of the 22 identified as women.

Additionally, the interviews were semi-structured and although this is a common approach it can impose interviewer bias, although it also designed to result in a coherent set of interviews which focus on common topics while still allowing interviewers to express their own views and experiences.

Finally, we did not attempt to verify claims that interviewees made about their classes, departments, or institutions, taking interviewee statements at face value.

5 ETHICS

Ethics of algorithms (including AI systems such as LLMs) is concerned with the societal context around algorithmic systems and how these systems affect both individuals and society. Our focus must therefore rest on aspects such as the provenance and quality of training data, the usage of AI systems, and associated costs (in the widest sense), rather than on how to "integrate ethics into the system" [41, 105]. Moreover, Mittelstadt et al. [140] point out that algorithmic systems tend to be large, complex and highly modularised, which makes it difficult in general to assign responsibilities. Hence, with an increasingly opaque training procedure of LLMs and a lack of clear responsibility in terms of authorship [172], the use of large language models naturally raises a number of concerns pertaining to academic integrity. Furthermore, using a large language model has been shown to affect the user's own opinions [102].

While a full discussion of the ethics of large language models in computing education is beyond the scope of this paper, we would like to highlight three crucial aspects: the role and stance of us as professionals in computing education, the policies brought forward by academic institutions, and the question of academic integrity in the context of large language models (which we will discuss in Section 6).

5.1 Ethics in LLM literature

In a study on the values encoded in the ML research literature, Birhane et al. identified a number of ethical values [45], from which the papers would draw motivation. In a total of 100 highly cited papers, the study found that *performance* was clearly the most frequent value and pointed out that performance is not a neutral term but comes with ethical implications. For instance, performance is typically measured with respect to specific benchmarks and data sets, thereby introducing bias—particularly when the dataset is thought to represent the "real world".

A full study of all values and their ethical implications found in the computing education literature is beyond the scope of this paper. However, we extracted explicitly stated motivations from the papers in our literature review as a first approximation to a full extraction and coding of values.

In line with Birhane et al. we observed that *performance*, *generalisation*, *efficiency*, *novelty*, and *scalability* were often mentioned. Many papers cited the "impressive" or "human-level" performance of current large language models. In contrast to what Birhane et al. report, the focus on performance in our dataset seems to be secondary as a means to highlight the timeliness of the research. This is particularly the case since the papers in our study did not propose performance improvements, but rather built on available performance. Furthermore, while performance in the ML research community typically relates to specific (and often well-known) benchmarks, we found that performance in the papers we reviewed usually referred to either the LLMs' ability to solve exercises, assignments or passing exams, or the LLMs' production of teaching materials, say, such as exercises.

With this focus on application of large language models rather than the design of a new AI system, we also found that several papers pointed out the potential to "save time" or help instructors

cope with growing class sizes. While these motivators may be understood as scalability issues, we believe that there is a difference in how this term would be understood in the paper surveyed by Birhane et al. Despite ostensible similarities concerning the underlying values encoded in the research literature, there might be some notable differences.

The free availability of large language models (i.e. that students can access LLMs at no cost) was a recurring theme in our surveyed literature. We would like to highlight this notion as an example of an ethically problematic assumption for two reasons. On the one hand, availability at "no cost" is often inaccurate because the costs might be hidden and paid, e.g., through provision of private data. On the other hand, ChatGPT offers a range of models with different performance characteristics, not all of which are free. Some students might therefore have access to more powerful tools than others. Investigating the assumptions, beliefs, and values held by the computing education community itself might therefore be well warranted. We call on the community to do so in future work.

5.2 Code of ethics implications

The IEEE Code of Ethics [9] comprises three sections. The first focuses on ethical standards, behaviour and conduct. The second section focuses on ethical treatment of other people, and the third focuses on compliance. The AAAI Code of Professional Ethics and Conduct [8] was adapted from the ACM Code of Ethics and has the same structure. The ACM Code of Ethics [10] comprises four sections. The first section outlines the fundamental ethical principles that all computing professions should use to guide thinking. Section 2 describes the ethical responsibilities of computing professionals, section 3 covers ethical leadership, and section 4 focuses on compliance. In the following sections, we use the ACM general ethical principles to frame the discussion of ethical issues raised by the use of large language models in computing education.

ACM General Ethical Principles. In this section we review policies from major universities around the world on the use of large language models in education. As of this writing, many universities do not currently have an official policy publicly available online. Instead, many universities are still presently working through the policy implications of large language models through task forces and other initiatives, such as at the University of Virginia [4], which is a top-ranked school for computer science in the USA. See Figure 2 for responses to the question "The policies at my university are clear regarding what is allowed and what is not allowed in terms of using GenAI tools", which illustrates this point.

To structure our review, we consider the first part of the ACM General Ethical Principles. Parts two through four of these principles were too specific for most university policies and therefore not as relevant to the present work. Most universities did not explicitly mention coding in their policies, with the exception of Yale University [13] and University of Adelaide [14].

To select universities for consideration of their LLM policies, we first found popular rankings of universities worldwide for computer science programs. Next, we examined policies at these universities, specifically those from Canada, USA, UK, and Australia: University of Toronto [7], Duke University [12], Yale University [13], Massachusetts Institute of Technology (MIT) [17], University of California Los Angeles (UCLA) [5], University of Adelaide [14], Monash University [18], and Oxford Brookes University [1]. We do not consider this a systematic attempt, nor would that be presently possible, given the conditions described above. Instead, this represents a purposeful sampling of top-ranked universities around the world to get an overall picture of how universities are responding to the appearance of LLMs.

The ACM General Ethical Principles are divided into the following sections:

Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing. Only MIT suggested that the arrival of LLMs into education is an opportunity to think about student academic well-being [17]. No universities sampled contained anything about general human well-being or the idea that we are all stakeholders in computing, or by extension, education or society in general. It seems the ideas in these policies are limited to the specific implications of using LLMs, rather than the general. We find this to be an unfortunate oversight because institutions can help guide students' concerns beyond themselves and their immediate circumstances to the broader collective human project of the pursuit of knowledge.

Avoid harm. Use of LLMs can lead to poor outcomes due to overreliance, ease of breaching academic integrity, and use of incorrect information leading to poor products. The universities we sampled all drew attention to the potential to create harm to others, institutions, and even the students themselves. Harm to others could come in the form of using the work of others without citation [12]. Harm to institutions can come in the form of students using LLM tools to generate work that is incorrect, offensive, or otherwise inappropriate and therefore dragging the university into potential altercation. It can also call into question the legitimacy of the learning outcomes and verification of them, which can jeopardise accreditation and institutional reputation or the community's trust in that institution.

Universities seemed most interested in helping students to understand the potential harm that students could incur upon themselves through inappropriate use of AI resources. Isserman writes that "plagiarism isn't a bad thing simply because it's an act of intellectual theft—although it is that. It's a bad thing because it takes the place of and prevents learning." [100]. One of the most significant concerns is the ease with which LLMs can produce the output that we ask students to produce. As educators, we are not interested in the output *per se*—rather we want students to engage in activities and processes that result in learning. Using a tool to produce the required output circumvents that learning, and deprives students from the opportunity to learn [100]. Many of the policies we sampled mentioned that students can prevent the growth of critical thinking skills and competencies in crucial areas by taking shortcuts through inappropriate use of these models [1, 5, 13, 18]. This is particularly important when writing code [13]. Not only does this harm their short-term ability to pass the course, but it also robs them of their long-term ability to master the subject matter and their preparedness for future work. Finally, harm to self, others, and institutions could come from inadequate understanding or preparation in courses where safety concerns are paramount, such as a chemistry lab. Skipping crucial learning could harm everyone

involved [14]. While most computing courses do not share similar safety concerns, it is possible in industry that inadequate learning or over-reliance on these tools could expose oneself, others, and institutions to harm (for example, by writing malformed code for self-driving cars).

Be honest and trustworthy. Most universities that we sampled wanted to make it clear that LLM tools are not reliable and can produce incorrect results. Duke warned faculty and students that LLM "output is only as good as its input" [12]. Others warned of the now well-known phenomenon of LLM "hallucination" where they will sometimes reply with incorrect responses when not enough is available [5] or create sources and facts even when enough data is available [7, 13, 18]. It could also be that generated content is incorrect simply because it uses data that is out of date [14] since many LLMs do not have access to data created after they were trained. It is clear that universities are thinking of honesty and trustworthiness in terms of the output of the tool, and when evaluated this way LLMs lack credibility.

Be fair and take action not to discriminate. Almost all university policies and guidelines made it a priority to discuss biases that exist in AI in general and LLMs in particular. Duke, UCLA, and Oxford Brookes pointed out that the models themselves often discriminate based on their training data, which means they receive all the stereotypes and misinformation from whatever data is used as input to the models [1, 5, 12]. Monash University warned faculty and students that LLMs may take data out of context, cannot predict future events with any amount of accuracy, and could present sensitive data inappropriately [18]. Finally, MIT noted that fairness could be at risk, based on who has access to the models [17]. While access to several popular models is currently free, other models require payment, so any advantage gained through the use of these models perpetuates existing social inequity.

Respect the work required to produce new ideas, inventions, creative works, and computing artefacts. University policies seemed to mostly skip this criterion. However, there were a few statements that are sufficiently related to warrant discussion here. For instance, Duke emphasised that creative writing or coding meant doing the hard work of developing each person's specific voice. Using AI tools could undercut that endeavour and cheat the student of their ability to develop innovative ideas, computing or otherwise [12]. This could be because, as Adelaide noted, AI tools lack originality and common sense [14]. Related to new ideas and innovation, UCLA noted in their policy that AI tools will often reflect outdated information that may fail to represent the progress of social movements since the training of the model was completed [5]. For example, if certain sets of rights were not legal when the model was trained, one should not expect the model to suggest that they should be.

Respect privacy. The lack of control over these tools, as well as what data they keep about their users, was mentioned in most policies. AI tools could invade users' privacy [12, 17], violate FERPA (a student privacy protection law in the United States) if student records are handed to it [5], do not assume that users are at least 18 years old [18], and are not bound by university ethics rules and policies. Furthermore, AI tools will often take user data and use it to train their models, whether users want that or not. Monash encouraged students and faculty to consider that their data could be stored by the model and used in other contexts [18].

Honor confidentiality. Ethical issues identified by universities included threats to confidentiality, though each one took a slightly different approach. Creating an account and using AI tools could bother students who are worried about the models stealing their intellectual property and may therefore be unwilling to use them [5]. The tools also do not respect the confidentiality of the people from whom they have taken the data to train the models, which could result in unintentional plagiarism for both students and faculty [18]. Finally, these models do not respect confidentiality with regard to legal issues, and data sent to them may be turned over to law enforcement agencies or other third-party vendors and affiliates without user consent [17].

6 ACADEMIC INTEGRITY IMPLICATIONS

Recent work on student use of generative AI tools has raised alarms about academic integrity violations [159]. Jones et al. summarises several common practices that are deemed to be cheating, distinguishing between plagiarism, collusion, and falsification [106]. We add contract cheating (as described by Deakin University [16]), and use of unauthorised resources (as described by University of Auckland [15]) to this list of practices that breach academic integrity. These terms are described by the respective documents as:

Plagiarism: A student incorporates another person's or body's work by unacknowledged quotation, paraphrase, imitation or other device in any work submitted for assessment in a way that suggests that it is the student's original work [106].

Collusion: The collaboration without official approval between two or more students (or between student(s) and another person(s)) in the presentation of work which is submitted as the work of a single student; or where a student(s) allows or permits their work to be incorporated in, or represented as, the work of another student [106].

Contract cheating: A student requests another person or service (including, according to Deakin University, artificial intelligence content production tools) to produce or complete all or part of an assessment task to submit as their own work [16].

Falsification: Where the content of any assessed work has been invented or falsely presented by the student [106].

Unauthorised resources: Using software, websites, materials or devices not explicitly permitted [15].

We discuss each of these academic integrity concerns with respect to generative AI.

6.1 Plagiarism

Plagiarism is the use of the work of others without appropriate attribution. This raises the issue of who is the author of work created by generative AI. There are several possibilities:

(1) The community that produced the source content used as input to the generative AI model is the author of the work.
(2) The generative AI software is the author of the work.
(3) The user of the generative AI software is the author of the work.

Although some in the literature treat the use of AI tools as plagiarism [144], we argue that although the community providing source material has influenced the generated content, this is similar to the natural process of writing in which authors read source material and use the information to generate new content, based on existing literature. Generative AI is almost always creating content *based on* the training data. In this case, the community has not authored the work generated by the model, so using AI-generated content would not be considered plagiarism of the original authors of work that was used as input to the generative AI model. In some rare cases, Generative AI tools can produce the work of someone else exactly, which would in fact be plagiarism. Since this is extremely rare, we do not consider it here other than to acknowledge it.

Although it may be tempting to consider generative AI to be the author of the work in all cases, academic publishers take an opposing view. Examples of statements include:

- "AI tools cannot meet the requirements for authorship as they cannot take responsibility for the submitted work. … Authors are fully responsible for the content of their manuscript, even those parts produced by an AI tool, and are thus liable for any breach of publication ethics." (Committee on Publication Ethics) [143]
- "AI does not meet the Cambridge requirements for authorship, given the need for accountability. AI and LLM tools may not be listed as an author on any scholarly work published by Cambridge." (Cambridge University Press) [2]
- "Authors should not list AI and AI-assisted technologies as an author or co-author, nor cite AI as an author. Authorship implies responsibilities and tasks that can only be attributed to and performed by humans." (Elsevier) [6]
- "Artificial Intelligence Generated Content (AIGC) tools—such as ChatGPT and others based on large language models (LLMs)—cannot be considered capable of initiating an original piece of research without direction by human authors. …—these tools cannot fulfil the role of, nor be listed as, an author of an article." (Wiley) [3]
- "Generative AI tools and technologies, such as ChatGPT, may not be listed as authors of an ACM published Work." (ACM) [11].

Our position is aligned with those of publishers that the *user* of generative AI tools should be considered the author of the work. This is consistent with the view of Pamela Samuelson, who states "The pragmatic answer to the AI authorship puzzle, …, the [author is the] user who is responsible for generating the outputs. If anyone needs to be designated as owner of rights in the outputs, it should be the user." [172] As such, we do not believe that the use of AI-generated content by students should be considered *plagiarism*, and it should not be referenced or cited as an independently authored piece of work. The student using the generative AI tool should be treated as the author of the work.

It should be noted that, in the apparent effort to combat plagiarism, there have been numerous tools that attempt to detect AI-generated material, such as CopyLeaks, GPTKit, GLTR, GPTZero, and AI writing detection from Turnitin. However, given the probabilistic nature of generative AI that is used to both generate the assignment in question and check the assignment, these tools are unreliable and produce many false positives [144]. Orenstrakh et al. found that these detectors are even worse when evaluating code [144].

6.2 Collusion

Collusion occurs when a student works together with another person to create work that they subsequently claim as their own. This requires both parties to willingly agree to work together and therefore assumes that both parties have agency. As generative AI has no agency, we do not consider a student who submits generated content to be engaged in collusion.

6.3 Contract cheating

Contract cheating is traditionally described as a student requesting another person to produce work that they submit as their own. The definition by Deakin University extends this view of contract cheating to explicitly include the use of generative AI tools [16] as do some other universities [18]. However, we disagree with this position, as it contradicts the view of the user of generative AI as the author—the position taken by academic publishers.

We recognise that generative AI models are capable of generating content that is more extensive than other software tools, but as a matter of principle, we consider the user to be the author (as discussed previously). Generative AI is a tool that may be used by a student to produce work, much as calculators and other software tools are used. We therefore do not believe that the use of generative AI software should be treated as contract cheating.

6.4 Falsification

Falsification occurs when a student invents or misrepresents data or results. The possibility that generative AI invents "facts" is well-known, and typically called *hallucination* [104]. Many university policies recognise that AI tools like LLMs can produce incorrect data and facts, which we discuss in Section 5.2 under the ACM requirement to "be honest and trustworthy." The view of publishers is that an author is responsible for the accuracy of the content in their work. Examples of policies from publishers include:

- "Authors are fully responsible for the content of their manuscript, even those parts produced by an AI tool, and are thus liable for any breach of publication ethics." (COPE) [143]
- "Where authors use generative AI and AI-assisted technologies in the writing process, these technologies should only be used to improve readability and language of the work. Applying the technology should be done with human oversight and control and authors should carefully review and edit the result, because AI can generate authoritative-sounding output that can be incorrect, incomplete or biased. The authors are ultimately responsible and accountable for the contents of the work." (Elsevier) [6]
- "The author is fully responsible for the accuracy of any information provided by the [generative AI] tool and for correctly referencing any supporting work on which that information depends." (Wiley) [3]
- "Authors are accountable for the accuracy, integrity and originality of their research papers, including for any use of AI." (Cambridge) [2]

We take the position that students who use generative AI are responsible for the content they include in their work. Inaccuracies, citations for non-existent papers, and other hallucinations that may arise from the use of generative AI are the responsibility of the student author. Therefore, the category of *Falsification* is a relevant academic integrity issue for students using generative AI. As discussed above, this could cause harm to students, anyone publishing their work, and to the university as a whole.

6.5 Use of unauthorised resources

Students are often required to engage in tasks that have specific constraints. For example, the use of calculators in general is acceptable, and we are comfortable with the notion that using a calculator for data analysis does not impact authorship, or normally breach academic integrity. However a student in a calculus class may be asked to solve a differential equation without the use of a calculator, and access to calculators may be restricted in secured assessments such as exams. Such constraints are typically imposed to ensure that learning outcomes are met (e.g., that the student can solve a differential equation without outside assistance). A student who used a calculator for a given assessment when it was not permitted could violate academic integrity for use of unauthorised resources.

Our position is that this is the appropriate category for the use of generative AI in computing education. Figure 3 shows responses to various ethical questions where it seems that many instructors and students agree that using generative AI tools to create an entire answer is wrong. In some courses, such as introductory programming, the use of generative AI may be undesirable as it can solve problems with minimal intellectual input from students. Not only could this violate academic integrity, but as discussed above, students who utilise these resources inappropriately in lower-level courses may cause harm to themselves by preventing their own preparation for upper-level coursework. However, in upper-level courses, it may be an appropriate productivity tool that students would be permitted to use, which the survey data in Figure 3 seems to corroborate given responses to questions about generating pieces of an assignment, help with style, or fixing bugs. However, this requires staff to be explicit in syllabi and/or assessment descriptions.

Advice for educators:

- We encourage educators to teach students about the appropriate ethical use of GAI throughout the curriculum and to allow the use of such tools where it is pedagogically appropriate. See Appendix D for an example.
- When educators assign assessed work for students, any restrictions on the use of tools such as generative AI should be explicitly stated.
- Students who use GenAI to complete assessed work should be required to include a statement about how it was used, consistent with academic publication requirements.
- Students using GenAI when they are not permitted, or in ways that are restricted, are engaged in misconduct by using unauthorised resources. Academic consequences as a result of this behaviour should be made clear in the course syllabus.

6.6 Advice for students

Simon et al. [183] highlights the importance of educating students about academic integrity and reveals a wide variety of ways that academic integrity is communicated to students. Given the disruptive nature of generative AI, we recommend that a guide for students be developed and distributed to students to provide explicit advice about the appropriate use of generative AI tools.

We recommend that any guidelines developed for students should:

- adopt professional practices and standards where possible; and
- adapt professional practices where needed to ensure good pedagogical practices are maintained.

Publishers typically require acknowledgement where generative AI has been used in the development of a manuscript. We believe this would be useful for teachers, and for students, to reflect on how generative AI was used, so we recommend that assessments require students to include a statement about how generative AI was used in assessment tasks (where permitted).

After considering the relevant findings from this report, we have developed a resource that provides guidance about the use of generative AI for students. It is by no means comprehensive or complete and reflects our perspectives on what students should know before using these tools. Our recommendations are informed by the risks identified in the literature, the ACM code of ethics, survey results, and the academic integrity documents that we analysed. We offer this to teachers as a resource that may be adapted and/or distributed to students.

Guide for students: See Appendix D for a sample handout or text that could be adapted and included in a course syllabus.

7 BENCHMARKING LARGE LANGUAGE MODELS FOR COMPUTING EDUCATION

In this section we focus on the performance of LLMs in the context of computing education. Teachers are interested in how well LLMs perform on tasks such as solving programming problems, explaining code, generating test questions, and providing feedback to students. However, the speed at which new models arrive and old models are deprecated is staggering. For example, the currently published literature which has focused on largely on GPT-3 may underestimate what the newest models, such as GPT-4, can do. Some recent work has found that GPT-4 outperforms earlier models in tasks such as visual programming [98, 184], Socratic questioning of novices [21], solving multiple-choice questions and programming exercises [175], and that performance can be close to human tutors for some tasks [151] while not for others [21].

In addition to being interested in how much the capabilities of LLMs in computing education related tasks have increased, we are interested in analysing the suitability of existing benchmarks for computing education as they might not translate to computing education settings. For instance, we expect that many students will be able to fix small mistakes made by LLMs themselves. Similarly, tasks in existing benchmarks might not match those typically found in computing education courses.

Another issue with current papers lies in our ability to validate results. A wide variety of parameters, prompts, and evaluation approaches have been used, and they are not always reported in detail. Furthermore, a slight variation in a prompt might generate quite different results. In this section we explore how we assess LLMs in the computing education context. We choose to focus on the task of generating a solution to a programming problem, because this is a major task for students and is a focus of existing literature.

First, we review datasets that are available for evaluating LLMs and tag problems in several of those datasets to assess where they fit in the context of computing education. Second, we take one of the first papers on LLMs that appeared in the computing education context [85] and replicate it using three more recent models. In the replication, we use the state-of-the-art GPT-4 model to give insight into how rapidly the performance of models has increased, the GPT-3.5-turbo model that powers the free version of ChatGPT which many students are likely to use, and GitHub Copilot which is free for students and educators and can be used as a plugin for popular IDEs. In addition, we openly release the problem descriptions and test cases for the dataset used in the original study [85] and our replication to facilitate future replication[6]. Finally, we report on our experiences running two analyses using openly available datasets, revealing the difficulties we encountered and their possible effects on results.

7.1 Review of empirical datasets

Table 7 presents a set of openly accessible datasets that have been or could be used to investigate questions about programming exercises or tasks in computing education contexts. To obtain the datasets presented in this table, we reviewed all of the papers in our literature review (previously presented in Tables 1 and 2) to identify any data that they used. We do not claim that this list is exhaustive, but it reflects the datasets in use when we conducted our literature review. In addition to these datasets, we are aware of one other attempt to review existing benchmarks for a particular task that might be performed by LLMs: natural language to code generation [197]. We believe the datasets listed in Table 7 represents a more broad set of applications and illustrates the kinds of questions being investigated and the breadth of educational contexts being examined.

A review of Table 7 suggests a number of limitations in the data available for pursuing LLM research in an educational context. Most of the datasets contain exercises (described in more or less structured natural language). This reflects a focus on the question of whether LLMs can solve typical programming problems. In contrast, relatively few datasets contain chat logs where students interact with an LLM or student-submitted code with syntax errors (for code repair tasks), but additional publicly available data for questions beyond code-generation would be beneficial, as that would allow researchers operating in smaller educational settings, where collecting sufficient data may be difficult, to engage in LLM work [116, 118].

Even for code generation tasks, where most of the data has been collected, it would be beneficial to collect additional data that is more diverse. Almost all of the datasets focus on Python (e.g., providing docstrings as input, Python starter or solution code, or student chats featuring Python code), with a smaller number of datasets featuring other languages, such as C/C++. Also, as described in the next section, many of the available datasets focus on small exercises used in introductory programming, with relatively few sources of data available to examine larger programming problems or content for more advanced courses.

Finally, as previously identified by Liu et al. [129], many of the published datasets do not provide robust evaluation of the exercises they include. Liu et al. [129] provide the EvalPlus dataset, which enhances previously published datasets with additional test cases; they found that the limited tests available meant that incorrect "solutions" were accepted as passing. We also found evaluation to be limited, with some datasets requiring manual intervention to complete evaluation. We provide more detail on issues we encountered when using these datasets in Section 7.4.

7.2 Analysis of problem types

To categorise the types of exercises present in the datasets, we manually analysed three datasets including HumanEval [56], FalconCode [64], and the data used in Finnie-Ansley et al.'s study [86]. The goal was to understand what kinds of problems the datasets included; whether they focused on introductory concepts or more advanced concepts such as object-oriented (OO) programming or data structures and algorithms. The results of these analyses are presented in Table 8.

HumanEval [56] and FalconCode [64] were tagged by a single author, an experienced instructor who has taught introductory programming, data structures, and advanced systems programming. The author tagged the exercises as being suitable for (a) an introductory programming course (Intro), as they use built-in data structures like strings and do not introduce complex algorithmic logic; (b) an introductory course using classes (OO), as they introduce classes or methods; or (c) a data structures or algorithms course, as they use some abstract data types (e.g., trees, queues, graphs) or complex algorithmic logic (e.g., sub-sequence matching, linear programming). The results are presented in Table 8. We found that in these two datasets, the vast majority of problems only cover material suitable in an introductory programming setting.

Many of the other datasets in Table 7 are similar to the two datasets we analysed, in that they appear to focus on introductory material. We requested access to the data used in Finnie-Ansley et al. [86]'s paper, as the topic was a more advanced (CS2) course. This time, the dataset was tagged by two authors (one who had tagged the previously discussed datasets and a second experienced instructor); we calculated Cohen's kappa and found near perfect agreement (0.94). Again, we found that the majority of the content was primarily suitable for an introductory course, with relatively few questions asking about object-oriented code or data structures. In addition, while reviewing the problems, we found that almost all were examples of small exercises, with relatively few requiring multiple functions to solve. The FalconCode [64] dataset includes a few exceptions to this general trend.

[6] osf.io/bu9h3/?view_only=a16f3e474be94188b884aa0dca02041f

Table 7: Prominent datasets available to evaluate LLM systems and tools.

Name	Description	Language	Size	Test cases	Solutions	Link
Mostly Basic Python Programs (MBPP) [28]	Natural language descriptions of introductory problems	Python	~1000	Yes	Yes	paperswithcode.com/dataset/mbpp
Search-Based Pseudocode-to-Code (SPOC) [119]	Pseudocode descriptions of coding problems	C++	18356	Yes	No	paperswithcode.com/dataset/spoc
Blackbox [49, 50]	Traces of editing and IDE interactions	Java	-	No	No	bluej.org/blackbox
Deepfix [91]	Student-generated code with syntax errors	C	6922	N/A	N/A	paperswithcode.com/dataset/deepfix
Automated Programming Progress Standard (APPS) [95]	Natural language descriptions of problems of various difficulties	Various	10000	Yes	Yes	paperswithcode.com/dataset/apps
HumanEval [56]	Docstring descriptions of (mostly introductory) programming problems	Python	164	Yes	No	paperswithcode.com/dataset/humaneval
Grounded CoPilot [32]	Programming tasks provided in an observation study	Python or Rust	4	No	No	github.com/michaelbjames/copilot-study
PyFiXV CodeForce [150]	Buggy submissions to programming contest problems for program repair	Python	240	No†	No	github.com/machine-teaching-group/edm2023_PyFiXV/tree/master
ChatGPT_Bioinformatics [152]	Natural language descriptions of introductory bioinformatics problems	Python	184	No	Partial	github.com/srp33/ChatGPT_Bioinformatics
EvalPlus [129]	Re-release of the HumanEval code-generation dataset with additional test-cases	Python	164	Yes	No	github.com/evalplus/evalplus
LeetCode Assistant [187]	LeetCode problem prompts, responses generated by LLMs, and attempts to repair buggy prompts	Python	1209	No†	No	zenodo.org/record/7792965#.ZCvyv-xBwUE
StudentEval [29]	Student-generated prompts for introductory programming problems	Python	1749	Yes	Yes	huggingface.co/datasets/wellesley-easel/StudentEval
FalconCode [64]	Natural language descriptions of introductory programming problems	Python	661	Yes	Yes	https://falconcode.dfcs-cloud.net/index.php
CS1QA [122]	Naturally-occurring questions asked by students to LLMs with the LLM response	Python	17698	N/A	N/A	github.com/cyoon47/CS1QA
AD2022 [148]	Problems with a data structures course with student solutions	Python	16	Yes	Yes	inf.uni-hamburg.de/en/inst/ab/lt/resources/data/adhrec
Digital TA [66]	Small exercises generated by a digital TA and associated solutions	Python	11	No	Yes	zenodo.org/records/7799972
Socratic Questioning [21]	Socratic dialogues with a human instructor	Python	86	N/A	N/A	aclanthology.org/2023.bea-1.57

† While test cases are not provided, the dataset consists of publicly available contest problems which have automated testing.

Table 8: Fraction of instructor-assigned tags (Introductory (Intro), Object Oriented (OO), or Data Structures (DS)) assigned to exercises in various datasets.

Dataset	Intro	OO	DS
HumanEval [56]	98.8%	0	1.2%
FalconCode [64]	100%	0	0
My AI CS2 [86]	83.3%	9.3%	7.4%

Finally, we examined the Automated Programming Progress Standard (APPS) dataset [95], as it explicitly advertises that it includes exercises of various difficulties. This is a large set (10000 problems), so we manually tagged a sample of 200 exercises and then used keyword searching to identify the usage of classes and common data structures. This means that our estimates will underestimate the complexity of the exercises. Many of the problems in this set *are* more complex, as expected as they were largely drawn from programming contests. However, they may not be problems typically seen in educational contexts. The instructor reviewing the problems would not use many of these problems in any course, as they introduce issues like floating point error, exceeding the maximum representable integer, or linear-time pattern searching. At the same time, relatively few explicitly reference OO topics (4.1%) or common data structures (6.6%).

Taken together, this analysis suggests that many of the available datasets—and as a result, the published results—reflect an early introductory programming context (CS1) with relatively few examples of common CS2 material (Cipriano and Alves [57] is a recent exception demonstrating research on OO topics) and even fewer covering any more advanced topics. Some datasets do include more challenging tasks, but these may not reflect the kinds of problems student programmers solve in upper year courses, as they appear to be inspired by (or were drawn directly from) programming contest sites.

7.3 Replication: The robots are coming

Finnie-Ansley et al. [85] published the first paper that examined the performance of large language models in solving introductory programming exercises. They found that Codex[7] had better performance than the median student on the same exam exercises. In order to understand how the performance of large language models in this task has improved in the past two years, we partially replicated their study.

Method: We contacted the authors of the original study and received the problem descriptions and test cases used in the study. The original study had a total of 30 exercises; 23 used in two exams and seven variants of the Rainfall-problem [87]. In the original study, the method used for the Rainfall-problems differed from the method used for the exam questions. In our replication, we follow the method originally used for the exam problems for both the exam and the Rainfall problems. We generate up to ten solutions for each problem, stopping if the LLM creates a solution that passes the tests, or includes a "trivial formatting error". As in the original article, we manually fixed the trivial errors and considered this step as an additional trial. All models were prompted with the problem description surrounded by triple-quotes as was done in the original study.

We evaluate three models in our replication: GPT-4, GPT-3.5-turbo, and GitHub Copilot. At the time of writing, GPT-4 is the state-of-the-art LLM, so it is interesting to see how it performs on solving the problems. GPT-4 also powers the paid version of ChatGPT. GPT-3.5-turbo is the model that powers the free version of ChatGPT, which is likely what many students will be using, and thus it is also interesting to include. Lastly, GitHub Copilot is free for students and educators, and is directly embedded into the IDE, so it is possible that students will be using it too, which is why we decided to include it in the replication.

Copilot works slightly differently to the other two models, as it cannot be directly 'prompted'. For evaluating Copilot, we used the Visual Studio Code Copilot plugin, and provided the prompt (problem description surrounded by triple-quotes) in an empty file. We would then wait for Copilot to provide a suggested completion, and would accept the first suggestion that Copilot provided. In the rare cases where no suggestion was provided based on just the problem description, we would write 'def' to start the function, after which Copilot would suggest code if it had not before.

Results (GPT-4): For GPT-4, we used a temperature value of 0.9 similar to the original study. The results of the replication are presented in Figure 4 (only GPT-4) and Table 9 (all three LLMs). It is clear that GPT-4 outperforms Codex, which is the model used in the original study. All problems were solved in under ten attempts, except for the last question in the second test (T2-Q12). As noted in the original study, some of the Rainfall variants had somewhat vague problem descriptions, leading to GPT-4 having more trivial formatting issues with the outputs. In the original study, the results of Codex were similar to students in the top quartile; in our replication, GPT-4 would have been one of the top students in the class. The only problem that GPT-4 was not able to solve was the last problem of the second test (T2-Q12), which involved drawing bar graphs using text where the shape of the bar graph was determined by values in a dict passed to the function.

Results (GPT-3.5): Similar to GPT-4, we used the temperature value of 0.9 for GPT-3.5. GPT-3.5 performed only slightly worse compared to GPT-4. It was able to solve most problems on the first try, although many solutions included trivial formatting issues (e.g., "print("The sum is", sum)" when the tests expected a period at the end of the string). Compared to GPT-4, GPT-3.5 could not solve one of the Rainfall variants, specifically the one from Simon. Looking into why GPT-3.5 struggled, the biggest issue for the model was that the problem description stated that "A day with negative rainfall is still counted as a day, but with a rainfall of zero." This was not taken into account in the code that GPT-3.5 generated as the code for all ten completions would simply ignore any days with negative rainfall values. Similar to GPT-4, GPT-3.5 could not solve the last question of the second test (T2-Q12).

Results (Copilot): Copilot performed the worst out of the three evaluated models, successfully solving 20 out of the 23 exam problems and four out of the seven rainfall variants. We looked into the issues in code for the problems that Copilot was unable to solve. T1-Q10 involved printing all words in a given sentence that start

[7] More specifically, the first version of the model 'code-davinci-001'.

with a given character. The problem description explicitly stated that "The sentence will end with a full-stop." The code generated by Copilot would not remove the full-stop from the end of the sentence, resulting in failure in the edge case when the last word of the sentence needed to be printed where the tests assumed the full-stop is not included in the word. T1-Q11 involved sorting four numbers given as a parameter using only the "min()" and "max()" functions (use of lists, if/elif/else, and loops was forbidden). While Copilot took the constraints into account, all ten completions had logical flaws and did not pass all the tests. Finally, similar to GPT-4 and GPT-3.5, Copilot was unable to solve T2-Q12 which involved printing bar graphs made of text. For T2-Q12, many completions from Copilot would not compile (the completion would be incomplete), and when it did, the code was nowhere near correct. For example, many completions would print the same bar graph regardless of input.

For the rainfall variants, Copilot was unable to correctly solve the one from Soloway, the one from Simon, and the one from Guzdial. For the variant from Simon, the issue was the same as for GPT-3.5 in that negative values were always ignored entirely, even though the problem description asked for these to be counted as days with a rainfall of 0. For both the Soloway variant and the variant from Guzdial, none of the completions generated by Copilot handled the edge case where the list is empty, leading to division by zero.

Copilot would sometimes provide just the function signature with a comment such as "# write your code here" followed by "pass", or the completion would not have any code, but only comments with suggestions/hints for how to start work on the problem.

Discussion: Overall, all models performed quite well, having performance that would have allowed them to pass the exams. Unsurprisingly, GPT-4 outperformed GPT-3.5 and Copilot, which corroborates previous work where GPT-4 has outperformed other LLMs for various tasks [21, 151, 175, 184]. The problems where any model struggled were either complex (e.g., printing bar graphs as text) or had vague problem descriptions, leading to some edge cases being ignored (e.g., rainfall variants that did not specify what should happen when the list is empty). As noted in the original study, LLMs struggle with trivial formatting issues, such as missing punctuation or producing extra output that is not specified by the problem description. We noticed less variation in the completions generated by Copilot compared to the ones from GPT-4 and GPT-3.5. This might be due to the use of a relatively high temperature value for these models—previous work found similar results and speculated that Copilot likely uses a lower temperature value [191].

Table 9: The replication results for [85]. An asterisk indicates that the solution required trivial modifications, which was counted as an additional attempt (i.e., "2*" means that the problem was solved on the first try, but had trivial mistakes e.g. in formatting of strings such as a missing period or extra unnecessary prints). A '-' means that the problem was not solved within 10 attempts.

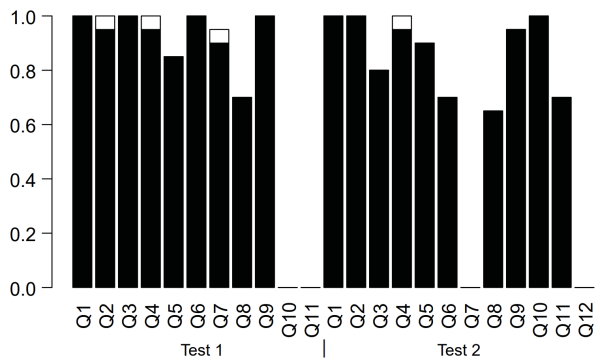
Problem	Solved on attempt		
	GPT-4	GPT-3.5	Copilot
T1-Q1	2*	1	2*
T1-Q2	1	1	1
T1-Q3	1	1	1
T1-Q4	2*	2*	2*
T1-Q5	1	2*	1
T1-Q6	1	2*	1
T1-Q7	2*	1	1
T1-Q8	1	1	1
T1-Q9	1	1	1
T1-Q10	1	1	-
T1-Q11	3	1	-
T2-Q1	1	2*	1
T2-Q2	1	2*	1
T2-Q3	1	1	1
T2-Q4	1	2	1
T2-Q5	1	2	4*
T2-Q6	1	1	1
T2-Q7	1	1	1
T2-Q8	2*	2	4
T2-Q9	1	1	1
T2-Q10	1	1	1
T2-Q11	2	1	1
T2-Q12	-	-	-
RF-Soloway	2*	2*	-
RF-Simon	2*	-	-
RF-Fisler	2*	2*	10
RF-Ebrahimi	2*	2*	2*
RF-Guzdial	2*	3	-
RF-Lakanen	2*	2	3*
RF-Apples	1	1	1

7.4 Novel analysis

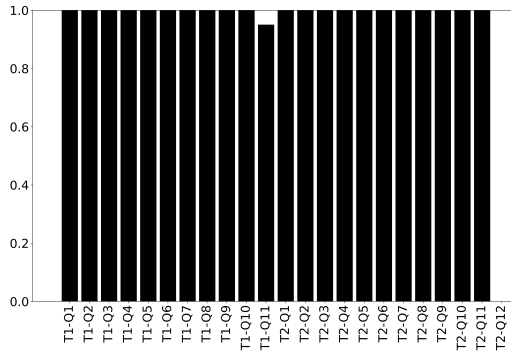
In addition to replicating the study by Finnie-Ansley et al. [85] (see Section 7.3), we analysed the performance of large language models in solving programming tasks found in two other datasets: the Automated Programming Progress Standard (APPS) dataset [95] and the FalconCode dataset [64]. We did not find existing evaluations of recent LLM performance on the programming tasks found in either dataset. Examining multiple datasets allows for more comprehensive evaluation of LLM performance.

7.4.1 APPS. Hendrycks et al. [95] created the Automated Programming Progress Standard (APPS) dataset as a benchmark for program generation. The dataset consists of 10,000 programming problems of varying difficulty, manually extracted from the online coding websites Codewars, AtCoder, Kattis, and Codeforces. The average number of lines for the solution is 18. The dataset has been divided into a training set and a test set, both containing 5000 problems. The following elements are provided for each problem:

- Problem description, including a description of the expected input and output of the problem and some concrete examples.
- A JSON file with inputs and corresponding output. On average, each problem has 21.2 test cases.
- A JSON file with metadata, with a difficulty level (introductory, intermediate, competition) and the url to the website where the problem is hosted.
- A list of solutions from humans.



(a) Results of the original "Robots Are Coming" paper that used Codex [85].



(b) Results of our replication of [85] with GPT-4.

Figure 4: A comparison of the original results and the score achieved by GPT-4 on the two CS1 tests and Rainfall-problem variants presented in [85].

Hendrycks et al. [95] tested the dataset in 2021 with GPT-2, GPT-3, and GPT-NEO. They found that GPT-NEO performed the best by passing almost 15% of test cases on introductory problems.

Method. We use the dataset from this paper and assess how newer models can handle these problems. We use a simplified method, employing the following steps:

We run the models (GPT-3.5-turbo16k, GPT-4) with the following parameters: temperature=0.0, max_tokens=4000, and default values for Top P (1), Frequency penalty (0), and Presence penalty (0). The system prompt we use is as follows:

```
You are a highly intelligent coding bot that can easily
handle any Python programming task. Given natural
language instructions you can always implement the
correct Python solution. Your focus is the solution
code only. You are not allowed to provide explanations.

Make sure to use input statements for input, and do not
give a method definition

Example (toy) instructions:
Implement a Python program to print "Hello, World!" in
the hello.py.

Example bot solution:
=== hello.py ===
x=input()
print(x)
===
```

As the user input, we provide the full problem description as described above.

We process the generated code by running the code and providing the inputs from the first 25 test cases to each solution. We then compare the output to the expected output using an exact comparison. After noticing that differences in characters for line endings caused problems ('\r\n' versus '\n'), we fixed this in the output comparison. We consider a test case to fail after a timeout of 5 seconds. For each problem, we store a list of test results and calculate a success rate as the percentage of passed test cases.

For simplicity, we skipped the problems that had starter code. We also skipped problem descriptions that could not be read. We ran a sample of 100 interview-level problems for GPT-4, running 25 test cases for each problem. We manually assessed the failing test cases, to check if the problems were actual coding problems or were caused by formatting mistakes. Minor issues were corrected.

The final runs were conducted in September 2023.

Results. Table 10 and Figure 5 show the results. GPT-4 performs quite well overall, with an average score of 51.5% on test cases. Comparing to GPT-3.5, which scored 39.2%, performance has clearly improved. Keeping a strict pass/fail criterion (all tests should pass), only 36.1% of the GPT-4 solutions pass all tests, as do 21.0% of GPT-3.5 solutions. We also observe a large difference between problem types, with GPT-4 solutions to introductory problems scoring as high as a 72.2% test case average, but the most difficult, competition level problems scoring only 28.7%.

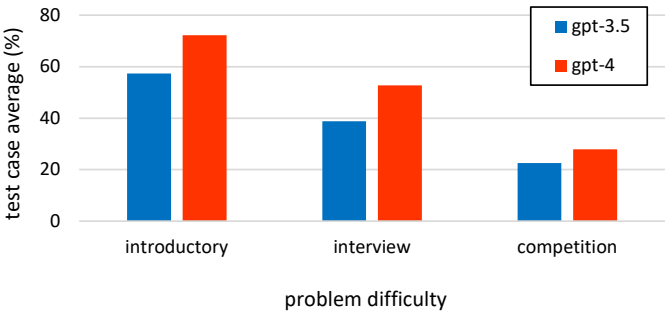


Figure 5: GPT success rate for different exercise types.

Discussion. A major downside of the APPS dataset is that it is a public dataset with problems from popular online coding websites.

Table 10: Average test case score for APPS problems run with max. 25 test cases.

Type	Count	Avg nr of tests	Test case average		Strict accuracy	
			GPT-3.5	GPT-4	GPT-3.5	GPT-4
Introductory	974	8.9	57.3%	72.2%	44.5%	62.9%
Interview	2972	15.3	38.8%	52.9%	18.7%	35.8%
Competition	1000	9.3	22.6%	28.7%	5.2%	10.8%
Overall	4946	12.9	39.2%	51.5%	21.0%	36.1%

It is possible that solutions for it have been included in training recent models.

Overall, APPS is a high-quality dataset with extensive test suites for many of the problems. However, in the context of computing education, several problems in this dataset might not be suitable for novice programmers. Even the introductory set contains some complex problems, and many are targeted at people participating in programming competitions.

We included an explicit instruction in the prompt to 'use input statements for input, and do not give a method definition.' Omitting this in a first test run showed many tests failing because the model returned a method definition. This shows that we need to be sure the model creates solutions in the exact same format as expected.

There are different aspects to be considered when running an analysis of a model on a certain task. The number of attempts to get a solution from the model should be specified. The 'Robots are coming'-replication (Section 7.3) performed multiple attempts, while we only use one attempt for this dataset.

7.4.2 FalconCode. FalconCode is a novel collection of over 1.5 million Python programs from over 2,000 undergraduate students at the United States Air Force Academy [64]. The dataset is not available online but is provided to anyone who applies following the instructions at the dataset website.[8] The dataset contains 661 introductory Python programming problems used in 4 courses. To understand how well selected LLMs (GPT-3.5/4) perform on these problems we (i) extracted the problem statements and unit tests; (ii) performed de-duplication of problem statements; (iii) utilised the LLMs to generate solutions; (iv) ran the unit tests provided in the dataset against the outputs of the LLMs; and (v) analysed the test results.

Data extraction. The dataset is provided in a convenient tabular format that has 661 problems. Among other fields, there are *prompt* and *test case* columns that contain the problem statement and the unit tests to evaluate the correctness of the solution. The problem statements are in HTML format which we preserved. The unit tests are runnable Python code. We extracted those into separate Python files.

Although the original dataset lists 661 problems, some problems are re-used across courses resulting in duplicate problem descriptions. After removing entries that have identical HTML problem descriptions, we were left with 385 unique problem descriptions.

Generating solutions. To generate a solution to the given problem, we include the problem statement (HTML) into the same prompt as used with the APPS dataset experiment (Subsection 7.4.1). We decided against extracting plain text of the problem statements because the HTML may encode important information through formatting that a state-of-the-art LLM such as GPT-3.5/4 may be capable of leveraging. Note that many of the problem statements refer to an external resource (e.g., starter code or a file with data) that has not been made available as part of the dataset. Hence, we cannot include the resource in the prompts submitted to the LLMs. We extract the completion of the submitted prompt and save it into a Python file named with the id of the problem (needed for the unit tests to discover the solution). In a non-negligible number of cases, the completion was wrapped in ```` ```python ...``` ```` tokens. While this issue could likely be fixed via further prompt engineering, we removed these tokens using a regular expression. Leaving the tokens in would cause syntax errors that would result in the failure of all the unit tests despite the solution being correct.

Testing. To test the correctness of the automatically generated solutions, we executed the provided unit tests. The unit tests rely on the solutions being present in the same directory in a file named with the id of the problem. Additionally, there is an external *cs110* grading (testing) library that needs to be installed through *pip*.[9]

We saved the output of the unit testing for each individual problem into a separate file. The last line of these had a predictable format: *Unit Test Returned: #*, where # is replaced with a number from 0 to 100. We utilised simple regular expressions to extract this final result of the evaluation from each of the output files. Note that assignments with identical problem statements could have been associated with different unit tests. Therefore, for each problem, we ran all available unit tests and took the average result of the tests as the score.

Results analysis. Table 11 shows the performance of GPT-3.5/4 on the 385 FalconCode problems across three different types of assignments:

- *Skill*: Small (1 – 3 line) programs focused on specific programming skill.
- *Lab*: Medium (10˜50 line) programs focused on utilisation of one or more skills.
- *Project*: Larger (50 – 300 line) programs solving an open-ended problem.

GPT-4 performed markedly better than GPT-3.5. In the subsequent analysis, we focus on the better performing GPT-4 model. The overall performance of 45.4% suggests a rather weak performance of the LLM in handling these introductory programming problems. In

[8] https://falconcode.dfcs-cloud.net/index.php

[9] https://pypi.org/project/cs110/

Table 11: Results for FalconCode. The middle column describes the raw success rates for each category of problem, and the rightmost column describes the success rates after problems where insufficient information is provided were removed.

	Full			Clean		
Type	**Count**	**GPT-3.5**	**GPT-4**	**Count**	**GPT-3.5**	**GPT-4**
Skill	162	30.3%	40.0%	81	57.6%	80.0%
Lab	214	29.8%	51.4%	155	47.5%	71.0%
Project	9	0.0%	0.0%	0	-	-
Overall	**385**	**29.3%**	**45.4%**	**236**	**51.4%**	**74.1%**

Table 12: Reasons for LLM (GPT-4) failures on FalconCode problems. We used this analysis to filter the dataset down to a clean version that is appropriate to use for the evaluation. The clean column signifies which reasons were considered as the failure on the LLM part. These data points were included in the clean dataset.

Failure Cause	**Clean**	**Skill**	**Lab**	**Project**
Missing Instructions				9
Missing Starter Code		58	7	
Missing Data File		20	46	
No Unit Tests		2	2	
Incorrect Unit Tests		1	4	
Unexpected Library	✓	1	2	
Incorrect Structure	✓		25	
Incorrect Solution	✓	22	28	
Overall Failed		104	114	9
Overall Failed (clean)		23	55	

order to understand the causes of the low performance we analysed each case where GPT-4 did not achieve the full score. Specifically, we performed a thematic analysis in which causes of each failed assignment were extracted as codes and then collated into higher-level themes [47]. The results of the analysis are shown in Table 12.

We first analysed the performance on Projects since it amounted to 0.0%. It turns out that the problem statement included in the dataset only points to an external pdf file that contains the actual instructions, e.g.:

> Objective: Create a drone simulation that can scan a battlefield for targets and engage them.
> Instructions: Read writeup (airstrike.pdf) and use the template file to begin work.

Since the pdf file has not been released with the dataset we could not provide the LLM with adequate instructions to generate a solution. This is the case for all 9 projects.

As Skill assignments are supposed to require only small solutions, not exceeding several lines of code, the performance of 40.0% is rather unexpected. Our analysis revealed that the main causes of the poor performance are related to the same cause detected with the Project tasks. There are a substantial number of situations where the Skill assignment required an external data file, and even more commonly starter code was needed to complete the assignment, e.g.:

> You have been provided with a list called list_of_animals. Write a program that prints out each of the items in this list (one item per line).

As the starter code has not been included in the dataset, the LLM does not have the complete information to produce the desired output. We also detected instances where the LLM used an unexpected library (not part of the Python standard library) and hence the program would crash, i.e., the unit tests would fail. In a few cases, the unit tests would be missing or incorrect. We also identified several instances where GPT-4 generated a genuinely incorrect solution to the problem statement that provided sufficient information.

The most common cause of failed unit tests for Lab assignments was also a missing data file and/or starter code (not released with the dataset). Another common cause was an incorrect structure of the (possibly correct) solution. A typical example would be a solution containing a function that returns a value whereas it was supposed to be a script asking a user to provide an input and print the output to the terminal. In the remaining cases GPT-4 generated a genuinely incorrect solution.

Based on the above analysis, we report another set of results (Clean) on the subset of 236 FalconCode problems that provide sufficient information for the LLM and are associated with valid test cases. The success rate increases from 45.4% to 74.1%. It is worth emphasising that if we would also disregard the cases where GPT-4 produced the correct solution using an unexpected structure (e.g., a function returning a value instead of a program asking user for an input and printing to a terminal) or utilised an unexpected library, the success rate would increase to 86.8%. Finally, a large portion of the genuinely incorrect solutions are rather superficial problems, such as not rounding to a single decimal as demonstrated in the example provided in the instructions. A simple change to the prompt adding the specific instruction would certainly fix such issues. Hence, one can conclude that we only observed a minimal amount of cases where GPT-4 would produce a truly incorrect solution to the problem (most certainly in fewer than 5% of cases).

7.5 Discussion

In this section, we provide an overview of the issues we encountered while performing our replication and analyses. Our experiences could provide valuable insights to researchers who want to study LLM performance as well as to teachers who are interested in their performance in an educational context.

Higher LLM performance than identified in the literature. Our replication of the Finnie-Ansley et al. [85] paper suggests that new LLM models are significantly more capable than is currently reported in the literature. Our experiment with the FalconCode dataset further supports this conclusion, and while success rates are lower on the APPS dataset, those problems are significantly more challenging than those typically provided in CS1 and CS2 courses.

Challenges using publicly available datasets. We encountered a number of challenges applying LLMs to publicly available datasets,

even though the datasets themselves are of high quality. In particular, we anticipate that future researchers will find that very few datasets will have been produced specifically to support LLM code generation research, so they are likely to not include critical information like starter code, data files, or formatting instructions.

Researchers will also encounter challenges even with datasets produced for code generation tasks. As suggested by Liu et al. [129], existing datasets may need to be augmented to evaluate LLM-generated code accurately. The number and quality of the test cases provided might not completely cover all exercise requirements and edge cases, therefore giving a false positive result. Alternately, test cases can even be incorrect, or too strict, exceeding what is required in the instructions, lowering the potential performance of models.

Homogeneity in available datasets. Finally, future researchers may struggle to find appropriate datasets. Most datasets we found could support code generation, using Python, of CS1 problems. To assess model performance on multiple types of tasks, for different programming languages, or at different levels, will require new datasets. The community will need to reward the effort of curating and maintaining such datasets, as providing a complete and well-evaluated dataset is challenging (as noted above) and is important for enabling research by a diverse set of groups.

7.5.1 Limitations. We focused our efforts on replicating code generation tasks, but there are many other research questions that are potentially even more challenging to replicate. Testing generated code can be easily automated by running test cases, although this might not capture all aspects relevant to computing educators, such as code quality and suitability of the solution with regards to which concepts the student has learned so far. These latter aspects could also be assessed automatically, but we have not attempted to do so in our study.

Outside of code generation, assessing solutions for other types of exercises common in CS (e.g. regular expressions, UML-diagrams, automata), LLM-generated exercises, feedback, and explanations require additional datasets and may require a qualitative framework that could be difficult to provide or to transfer to another research team.

Advice for users and creators of LLM Computing Education datasets:

- Creators: Include full and precise problem descriptions, so that the LLM can be given sufficient information for solving the problem.
- Creators: Include full test cases, ideally in a format where they are easy to run for others, so that LLM performance can be easily evaluated.
- Creators: Include any resources needed to complete the assignments, e.g., the starter code or data files.
- Creators: Make it easy to update or extend the data set, and report issues.
- Users: Make LLM parameters clear for replication.
- Users: Clearly describe the prompts used with the models, ideally providing example prompts.

8 CONCLUSIONS

This report is the output of an ITiCSE Working Group that explored how the emerging generative AI revolution will impact the future of computing education. The first time that the group met was in April 2023—three years after the release of the ground-breaking GPT-3 large language model; less than two years after the release of the Codex model (a variant of GPT-3 specifically fine-tuned for coding tasks); less than one year since the Copilot plug-in (for generating code directly within an IDE) was made available for free to students worldwide; five months since the release of ChatGPT (providing a convenient chatbot interface); and just one month after the release of GPT-4, a powerful multi-modal large language model. Against this backdrop of rapid advancements, our working group came together at a time when the computing education community was just beginning to grapple with the widespread use of generative AI tools by students as well as the general public. Many urgent questions were being asked about how to adapt to the challenges and opportunities presented by these new models and tools. In particular, if students are able to generate solutions to all of their programming coursework, how will this impact what is taught, how it is taught, and how students will remain motivated to learn?

Our overarching goal is for this report to serve as a focal point for researchers and practitioners who are exploring, adapting, using, and evaluating LLMs and LLM-based tools in computing classrooms. We now return to the list outlined in Section 1.1 to summarise our main contributions:

(1) **A review of the literature:** We provide a detailed review of the literature on LLMs in computing education, current as of August 2023. Using a keyword search of relevant databases and two rounds of forward and backward snowballing, we synthesise findings from 71 primary articles. Due to rapid changes in the field and the slow pace of publishing in traditional venues, much of this work was available only as pre-prints on platforms such as arXiv. We included all such literature in our review and assessed every article with respect to a set of quality metrics. The most common type of paper to date involves evaluating the performance of LLMs when applied to tasks such as solving programming exercises. A key finding, which justifies some of the widely voiced concerns around academic misuse of LLMs, is that current models tend to perform at least as well as most students on typical introductory-level programming tasks. We also reviewed papers that discussed possible opportunities and challenges of LLMs, that studied how end-users (including students) interacted with LLMs, and that used LLMs to generate high-quality learning resources. Among the risks that were identified, the most common concern expressed by authors was that students would become overly-reliant on using LLMs to generate and debug code.
(2) **Prevailing attitudes:** To understand how LLMs are currently being perceived and used, we conducted a survey involving 171 students and 57 instructors from computing courses spanning 20 distinct countries. We found that, in general, students and instructors had similar perceptions about LLMs with respect to questions around experiences, expectations and beliefs. However, they differed in their perceptions of how clear course

policies were about the allowed use of LLMs, with instructors—somewhat surprisingly—finding these policies to be less clear than students. Many of the respondents to our survey had very little experience using generative AI tools at the current time, although we expect familiarity to grow rapidly in the coming years. Some instructors were concerned that their students were using such tools inappropriately, and a small fraction of students refused to use generative AI tools for ethical reasons and due to concerns about harming their learning. In many cases perceptions were well-aligned—both students and instructors felt strongly that there should be some restrictions on the allowed usage of generative AI tools for coursework.

(3) **New instructional approaches:** Although many instructors are only just beginning to think about the impacts on their teaching, some have already made concrete changes to their curricula and assessments. In order to document these recent adaptations, we conducted 22 in-depth interviews with instructors on five continents who already had concrete plans in place to change some aspect of their teaching. We found that some instructors were beginning to place a greater emphasis on 'process over product'. That is, instead of just grading a final artefact, there is an evaluation of the processes used by students when working on a product. In addition, there was also a trend towards placing a greater emphasis on invigilated assessments such as exams, with a reduction to the grade weighting placed on unsupervised homework assignments. We anticipate further changes to learning objectives, course content and assessment practices in the near future, and we see an important need to swiftly disseminate best practices that emerge.

(4) **Academic integrity: Policies & recommendations** We reviewed academic integrity policies that mentioned generative AI from major universities around the world and found that these explicitly addressed many of the principles stated in the ACM code of ethics. However, it is unclear how students are being educated about the ethical use of generative AI in the classroom. This appears to be an important area for future work given the findings from our survey which revealed that students and instructors have quite different views regarding the clarity of current policies. Further work in this area is needed to understand how to effectively embed these principles in computing classrooms.

We follow our review of policies with concrete recommendations for both students and instructors. We agree with the position articulated by many publishers that the user of the LLM should be considered the author of the generated text. This has implications for academic integrity, in that *plagiarism* is not usually a concern but instead users would be responsible for any *falsification* produced from uncritical use of LLM-generated artefacts. We encourage instructors to teach students about the ethical use of generative AI throughout their courses, clearly stipulating any restrictions on use for assessed work. Should students use such tools for graded tasks, we recommend they include a statement detailing its usage, and any violations should be regarded as academic misconduct with the penalties explained clearly. Institutions and faculty will need to communicate these expectations explicitly, and thus it is imperative that we provide students with resources to understand how to use LLMs appropriately. To this end, we have prepared a sample handout that can be adapted and included in a course syllabus on the ethics of using generative AI tools for assignment work (see Appendix D).

(5) **Encouraging replication:** Given that instructors are naturally interested in how well LLMs can solve typical tasks that are set for students, a common thread of work to date has been to evaluate the performance of various models. However, replicating prior work using newer models is difficult, given that a wide variety of parameters, prompts, and evaluation approaches have been used, and not all methods are reported with sufficient detail. Producing a dataset that contains everything necessary for high-quality LLM research (in particular, accurate evaluation of the artefacts generated by LLMs) is challenging and needs to be encouraged by the community. We therefore identified a seminal paper on LLM evaluation for programming tasks [85], and have prepared and released the problem descriptions and test cases in order to facilitate future replication work. Our own replication of this prior work, using a state-of-the-art model, shows an extraordinary performance improvement over the span of two years since the original work was carried out.

As we face the changes being ushered in by the AI revolution, it is clear that LLMs present significant challenges but also new opportunities for computing educators. We present this report not only as a snapshot of the current state at this relatively early stage, but also as a call to action: to encourage broad exploration of the use and impacts of Generative AI and LLM-based tools in computing classrooms, to adapt teaching methods and update academic integrity policies, and to develop best practices and to share them widely with the computing education community. Many pressing matters presently remain unknown. For instance, future work must explore how these new tools impact equity, justice, and diversity in computing education. Will these generative AI tools have a positive impact on marginalised groups or will it widen the gap? The future of computing education is rapidly evolving, and shaping it towards the common good must be a collective effort.

ACKNOWLEDGMENTS

Thank you to the following:

- Michael Caspersen for joining this effort in the early days and graciously bowing out when more prestigious matters intervened;
- All of the students and educators who responded our surveys;
- All of the educators who participated in interviews (in alphabetical order by surname);
  - Austin Cory Bart (University of Delaware, USA)
  - Michael Caspersen (It-vest & Aarhus University, Denmark)
  - James Davenport (University of Bath, UK)
  - Rodrigo Duran (Federal Institute of Mato Grosso do Sul Brazil)
  - Dan Garcia (UC Berkeley, USA)
  - Michael Kölling (King's College London, UK)
  - Viraj Kumar (Indian Institute of Science, Bengaluru, India)
  - Mark Liffiton (Illinois Wesleyan University, USA)
  - Jérémie Lumbroso (University of Pennsylvania, USA)

– Peter Mawhorter (Wellesley College, USA)
– Jean Mehta (Saint Xavier University, USA)
– Briana Morrison (University of Virginia, USA)
– Leo Porter (University of California San Diego, USA)
– Jan Schneider (Goethe University, Germany)
– David H. Smith IV (University of Illinois, Urbana-Champaign, USA)
– Kristin Stephens-Martinez (Duke University, USA)
– Sven Strickroth (LMU Munich, Germany)
– Ewan Tempero (University of Auckland, New Zealand)
– Christian Tomaschitz (TU Wien, Austria)
– Frank Vahid (University of California, Riverside, USA)
– those who wished to be de-identified in this report

- Juho Leinonen would like to thank the Ulla Tuominen Foundation.

REFERENCES

[1] [n. d.]. Artificial Intelligence. https://www.brookes.ac.uk/students/academic-development/online-resources/Artificial-intelligence. Accessed: 06 September 2023.
[2] [n. d.]. Authorship and Contributorship. https://www.cambridge.org/core/services/authors/publishing-ethics/research-publishing-ethics-guidelines-for-journals/authorship-and-contributorship#ai-contributions-to-research-content Accessed: 08 July 2023.
[3] [n. d.]. Best Practice Guidelines on Research Integrity and Publishing Ethics. https://authorservices.wiley.com/ethics-guidelines/index.html#5 Accessed: 08 July 2023.
[4] [n. d.]. Generative AI in Teaching and Learning Task Force (GENAI). https://provost.virginia.edu/subsite/genai. Accessed: 08 July 2023.
[5] [n. d.]. Guidance For the Use of Generative AI. https://teaching.ucla.edu/resources/ai_guidance/. Accessed: 08 July 2023.
[6] [n. d.]. Publishing Ethics | Elsevier Policy. https://beta.elsevier.com/about/policies-and-standards/publishing-ethics?trial=true#4-duties-of-authors Accessed: 08 July 2023.
[7] [n. d.]. Using ChatGPT or Other Generative AI Tool on a Marked Assessment. https://www.academicintegrity.utoronto.ca/perils-and-pitfalls/using-chatgpt-or-other-ai-tool-on-a-marked-assessment/. Accessed: 06 September 2023.
[8] 2019. AAAI Code of Professional Ethics and Conduct. https://aaai.org/about-aaai/ethics-and-diversity/. Accessed: 08 July 2023.
[9] 2020. IEEE Code of Ethics. https://www.ieee.org/about/corporate/governance/p7-8.html Accessed: 08 July 2023.
[10] 2023. ACM Code of Ethics. https://www.acm.org/code-of-ethics Accessed: 08 July 2023.
[11] 2023. ACM Policy on Authorship. https://www.acm.org/publications/policies/new-acm-policy-on-authorship Accessed: 08 July 2023.
[12] 2023. AI and Teaching at Duke. https://learninginnovation.duke.edu/ai-and-teaching-at-duke/. Accessed: 08 July 2023.
[13] 2023. AI Guidance. https://poorvucenter.yale.edu/AIguidance. Accessed: 08 July 2023.
[14] 2023. Artificial Intelligence. https://libguides.adelaide.edu.au/c.php?g=959585&p=6965069. Accessed: 08 July 2023.
[15] 2023. Student Academic Conduct Statute. https://cdn.auckland.ac.nz/assets/auckland/about-us/about-the-university/policy-hub/Student%20Academic%20Conduct%20Statute%20-%20approved%20by%20Council%202023-03-14(1).pdf Accessed: 08 July 2023.
[16] 2023. Student Academic Integrity Policy. https://policy.deakin.edu.au/document/view-current.php?id=107&_ga=2.136370873.847668142.1687377795-888769290.1685566682 Accessed: 08 July 2023.
[17] 2023. Teaching & Learning with ChatGPT: Opportunity or Quagmire? Part III. https://tll.mit.edu/teaching-learning-with-chatgpt-opportunity-or-quagmire-part-iii/. Accessed: 08 July 2023.
[18] 2023. Using Artificial Intelligence. https://www.monash.edu/learnhq/build-digital-capabilities/create-online/using-artificial-intelligence. Accessed: 08 July 2023.
[19] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013.* Technical Report. ACM Press and IEEE Computer Society Press. https://doi.org/10.1145/2534860
[20] Toufique Ahmed, Noah Rose Ledesma, and Premkumar Devanbu. 2023. SynShine: Improved Fixing of Syntax Errors. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2169–2181. https://doi.org/10.1109/TSE.2022.3212635
[21] Erfan Al-Hossami, Razvan Bunescu, Ryan Teehan, Laurel Powell, Khyati Mahajan, and Mohsen Dorodchi. 2023. Socratic Questioning of Novice Debuggers: A Benchmark Dataset and Preliminary Evaluations. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*. 709–726.
[22] 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
[23] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An Analysis of Using Many Small Programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. ACM, NY, NY, USA, 585–591. https://doi.org/10.1145/3287324.3287466
[24] Pedro Alves and Bruno Pereira Cipriano. 2023. The Centaur Programmer - How Kasparov's Advanced Chess Spans Over to the Software Development of the Future. arXiv:2304.11172 [cs.HC]
[25] 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]
[26] Mikko Apiola, Sonsoles López-Pernas, and Mohammed Saqr. 2023. *Past, Present and Future of Computing Education Research: A Global Perspective.* Springer Nature.
[27] Lena Armstrong, Jayne Everson, and Amy J. Ko. 2023. Navigating a Black Box: Students' Experiences and Perceptions of Automated Hiring. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1* (Chicago, IL, USA) *(ICER '23 V1)*. Association for Computing Machinery, New York, NY, USA, 148–158. https://doi.org/10.1145/3568813.3600123
[28] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program Synthesis With Large Language Models. *arXiv preprint arXiv:2108.07732* (2021).
[29] Hannah McLean Babe, Sydney Nguyen, Yangtian Zi, Arjun Guha, Molly Q Feldman, and Carolyn Jane Anderson. 2023. StudentEval: A Benchmark of Student-Written Prompts for Large Language Models of Code. *arXiv preprint arXiv:2306.04556* (2023).
[30] Rishabh Balse, Bharath Valaboju, Shreya Singhal, Jayakrishnan Madathil Warriem, and Prajish Prasad. 2023. Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 292–298. https://doi.org/10.1145/3587102.3588852
[31] Yeting Bao and Hadi Hosseini. 2023. Mind the Gap: The Illusion of Skill Acquisition in Computational Thinking. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 778–784. https://doi.org/10.1145/3545945.3569749
[32] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 78 (apr 2023), 27 pages. https://doi.org/10.1145/3586030
[33] Brett Becker, James Prather, Paul Denny, Andrew Luxton-Reilly, James Finnie-Ansley, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation. In *Proceedings of the 54th SIGCSE Technical Symposium on Computer Science Education* (Toronto, Canada) *(SIGCSE '23)*. ACM.
[34] Brett A. Becker. 2017. Artificial Intelligence in Education: What Is It, Where Is It Now, Where Is It Going. *Ireland's Yearbook of Education* 2018 (2017), 42–46.
[35] Brett A. Becker. 2021. What Does Saying That 'Programming is Hard' Really Say, and about Whom? *Commun. ACM* 64, 8 (jul 2021), 27–29. https://doi.org/10.1145/3469115
[36] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE-WGR '19)*. ACM, NY, NY,USA, 177–210. https://doi.org/10.1145/3344429.3372508
[37] Brett A. Becker and Thomas Fitzpatrick. 2019. What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 1011–1017. https://doi.org/10.1145/3287324.3287485
[38] Brett A. Becker and Keith Quille. 2019. 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 338–344. https://doi.org/10.1145/3287324.3287432

[39] 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 *Proceedings of the 15th International Conference on Computer Supported Education - Volume 2: CSEDU*. INSTICC, SciTePress, 59–69. https://doi.org/10.5220/0012007500003470

[40] Mordechai Ben-Ari. 2001. Constructivism in Computer Science Education. *Journal of computers in Mathematics and Science Teaching* 20, 1 (2001), 45–73.

[41] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (Virtual Event, Canada) *(FAccT '21)*. Association for Computing Machinery, New York, NY, USA, 610–623. https://doi.org/10.1145/3442188.3445922

[42] John Biggs. 1996. Enhancing Teaching Through Constructive Alignment. *Higher education* 32, 3 (1996), 347–364.

[43] John B Biggs and Kevin F Collis. 2014. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press.

[44] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking Flight with Copilot: Early Insights and Opportunities of AI-Powered Pair-Programming Tools. *Queue* 20, 6 (jan 2023), 35–57. https://doi.org/10.1145/3582083

[45] Abeba Birhane, Pratyusha Kalluri, Dallas Card, William Agnew, Ravit Dotan, and Michelle Bao. 2022. The Values Encoded in Machine Learning Research. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 173–184. https://doi.org/10.1145/3531146.3533083

[46] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2022. On the Opportunities and Risks of Foundation Models. arXiv:2108.07258 [cs.LG]

[47] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[48] Robert W Brennan and Jonathan Lesage. 2022. Exploring the Implications of Openai Codex on Education for Industry 4.0. In *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer, 254–266.

[49] Neil C. C. Brown, Amjad Altadmri, Sue Sentance, and Michael Kölling. 2018. Blackbox, Five Years On: An Evaluation of a Large-Scale Programming Data Collection Project. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 196–204. https://doi.org/10.1145/3230977.3230991

[50] Neil C. C. Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 223–228. https://doi.org/10.1145/2538862.2538924

[51] Peter Brusilovsky, Barbara J. Ericson, Cay S. Horstmann, Christian Servin, Frank Vahid, and Craig Zilles. 2023. The Future of Computing Education Materials. https://csed.acm.org/wp-content/uploads/2023/03/Educational-Materials-First-Draft-1.pdf First Draft, to be published in the CS2023: ACM/IEEE-CS/AAAI Computer Science Curricula.

[52] Christopher Bull and Ahmed Kharrufa. 2023. Generative AI Assistants in Software Development Education: A vision for integrating Generative AI into Educational Practice, Not Instinctively Defending Against it. *IEEE Software* (2023).

[53] Cecilia Ka Yuk Chan. 2023. A Comprehensive AI Policy Education Framework for University Teaching and Learning. arXiv:2305.00280 [cs.CY]

[54] 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]

[55] Cecilia Ka Yuk Chan and Louisa H. Y. Tsi. 2023. The AI Revolution in Education: Will AI Replace or Assist Teachers in Higher Education? arXiv:2305.01185 [cs.CY]

[56] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[57] Bruno Pereira Cipriano and Pedro Alves. 2023. GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 61–67. https://doi.org/10.1145/3587102.3588814

[58] Victoria Clarke and Virginia Braun. 2021. Thematic Analysis: A Practical Guide. *Thematic Analysis* (2021), 1–100.

[59] Alison Clear, Allen Parrish, John Impagliazzo, Pearl Wang, Paolo Ciancarini, Ernesto Cuadros-Vargas, Stephen Frezza, Judith Gal-Ezer, Arnold Pears, Shingo Takada, Heikki Topi, Gerrit van der Veer, Abhijat Vichare, Les Waguespack, and Ming Zhang. 2020. *Computing Curricula 2020 Paradigms for Global Computing Education*. ACM, New York.

[60] Stephen Cooper, Wanda Dann, and Randy Pausch. 2003. Teaching Objects-First in Introductory Computer Science. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Navada, USA) *(SIGCSE '03)*. Association for Computing Machinery, New York, NY, USA, 191–195. https://doi.org/10.1145/611892.611966

[61] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jiang. 2023. Github Copilot AI Pair Programmer: Asset or Liability? *Journal of Systems and Software* (2023), 111734.

[62] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, Zhen Ming, and Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? arXiv:2206.15331 [cs.SE]

[63] Nell B. Dale. 2006. Most Difficult Topics in CS1: Results of an Online Survey of Educators. *SIGCSE Bull.* 38, 2 (jun 2006), 49–53. https://doi.org/10.1145/1138403.1138432

[64] Adrian de Freitas, Joel Coffman, Michelle de Freitas, Justin Wilson, and Troy Weingart. 2023. FalconCode: A Multiyear Dataset of Python Code Samples from an Introductory Computer Science Course. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 938–944. https://doi.org/10.1145/3545945.3569822

[65] DeepLearning.AI. 2023. Generative AI with LLMs. https://www.deeplearning.ai/courses/generative-ai-with-llms/

[66] Liliya A Demidova, Elena G Andrianova, Peter N Sovietov, and Artyom V Gorchakov. 2023. Dataset of Program Source Codes Solving Unique Programming Exercises Generated by Digital Teaching Assistant. *Data* 8, 6 (2023), 109.

[67] Paul Denny, Brett A. Becker, Michelle Craig, Greg Wilson, and Piotr Banaszkiewicz. 2019. Research This! Questions That Computing Educators Most Want Computing Education Researchers to Answer. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 259–267. https://doi.org/10.1145/3291279.3339402

[68] Paul Denny, Brett A. Becker, Juho Leinonen, and James Prather. 2023. 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* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, NY, USA, 3–4. https://doi.org/10.1145/3587102.3588773 Video: www.youtube.com/watch?v=KwVcRXQc3IU.

[69] 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]

[70] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1136–1142. https://doi.org/10.1145/3545945.

3569823
[71] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett Becker, and Brent Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR USA) *(SIGCSE '24)*. ACM.
[72] 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]
[73] Paul Denny, James Prather, Brett A. Becker, Zachary Albrecht, Dastyni Loksa, and Raymond Pettit. 2019. A Closer Look at Metacognitive Scaffolding: Solving Test Cases Before Programming. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '19)*. Association for Computing Machinery, New York, NY, USA, Article 11, 10 pages. https://doi.org/10.1145/3364510.3366170
[74] 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]
[75] Paul Denny, Sami Sarsa, Arto Hellas, and Juho Leinonen. 2022. Robosourcing Educational Resources – Leveraging Large Language Models for Learnersourcing. https://doi.org/10.48550/ARXIV.2211.04715
[76] Paul E. Dickson, Neil C. C. Brown, and Brett A. Becker. 2020. Engage Against the Machine: Rise of the Notional Machines as Effective Pedagogical Devices. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 159–165. https://doi.org/10.1145/3341525.3387404
[77] Catherine D'ignazio and Lauren F Klein. 2020. *Data Feminism.* MIT press.
[78] Felix Dobslaw and Peter Bergh. 2023. Experiences with Remote Examination Formats in Light of GPT-4. *arXiv preprint arXiv:2305.02198* (2023).
[79] Felix Dobslaw and Peter Bergh. 2023. Experiences with Remote Examination Formats in Light of GPT-4. In *Proceedings of the 5th European Conference on Software Engineering Education* (Seeon/Bavaria, Germany) *(ECSEE '23)*. Association for Computing Machinery, New York, NY, USA, 220–225. https://doi.org/10.1145/3593663.3593695
[80] Augie Doebling and Ayaan M. Kazerouni. 2021. Patterns of Academic Help-Seeking in Undergraduate Computing Students. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research* (Joensuu, Finland) *(Koli Calling '21)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/3488042.3488052
[81] Thomas Dohmke, Marco Iansiti, and Greg Richards. 2023. Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle. arXiv:2306.15033 [econ.GN]
[82] Stefania Druga and Nancy Otero. 2023. Scratch Copilot Evaluation: Assessing AI-Assisted Creative Coding for Families. arXiv:2305.10417 [cs.HC]
[83] Joseph V. Elarde and Fatt-Fei Chong. 2011. Introductory Computing Course Content: Educator and Student Perspectives. In *Proceedings of the 2011 Conference on Information Technology Education* (West Point, New York, USA) *(SIGITE '11)*. Association for Computing Machinery, New York, NY, USA, 55–60. https://doi.org/10.1145/2047594.2047610
[84] Neil A. Ernst and Gabriele Bavota. 2022. AI-Driven Development Is Here: Should You Worry? *IEEE Software* 39, 2 (mar 2022), 106–110. https://doi.org/10.1109/ms.2021.3133805
[85] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/3511861.3511863
[86] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know If This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE '23)*. ACM, NY, NY, USA, 97–104. https://doi.org/10.1145/3576123.3576134
[87] Kathi Fisler. 2014. The Recurring Rainfall Problem. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (Glasgow, Scotland, United Kingdom) *(ICER '14)*. ACM, NY, NY, USA, 35–42. https://doi.org/10.1145/2632320.2632346
[88] Norbert Forman, József Udvaros, and Mihály Szilárd Avornicului. 2023. ChatGPT: A New Study Tool Shaping the Future for High School Students. *International Journal of Advanced Natural Sciences and Engineering Researches* 7, 4 (May 2023), 95–102. https://doi.org/10.59287/ijanser.562
[89] Fiona French, David Levi, Csaba Maczo, Aiste Simonaityte, Stefanos Triantafyllidis, and Gergo Varda. 2023. Creative Use of OpenAI in Education: Case Studies from Game Development. *Multimodal Technologies and Interaction* 7, 8 (2023). https://doi.org/10.3390/mti7080081
[90] Pavel Gherciu. 2022. Net Impact of Large Language Models Trained on Code. In *Conferința tehnico-științifică a studenților, masteranzilor și doctoranzilor*, Vol. 1. 189–192.
[91] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. Deepfix: Fixing Common C Language Errors by Deep Learning. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 31.
[92] Arto Hellas, Petri Ihantola, Andrew Petersen, Vangel V. Ajanovski, Mirela Gutica, Timo Hynninen, Antti Knutas, Juho Leinonen, Chris Messom, and Soohyun Nam Liao. 2018. Predicting Academic Performance: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. Association for Computing Machinery, New York, NY, USA, 175–199. https://doi.org/10.1145/3293881.3295783
[93] 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).
[94] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 93–105. https://doi.org/10.1145/3568813.3600139
[95] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence With APPS. *NeurIPS* (2021).
[96] Wayne Holmes. 2023. Special Issue on Artificial Intelligence in Education: Coming of Age? *International Journal of Artificial Intelligence in Education* (2023), 1–11. https://link.springer.com/collections/igedgdciea
[97] Wayne Holmes, Maya Bialik, and Charles Fadel. 2023. *Artificial Intelligence in Education.* Globethics Publications. https://doi.org/10.58863/20.500.12424/4276068
[98] Irene Hou, Owen Man, Sophie Mettille, Sebastian Gutierrez, Kenneth Angelikas, and Stephen MacNeil. 2023. More Robots are Coming: Large Multimodal Models (ChatGPT) can Solve Visually Diverse Images of Parsons Problems. *arXiv preprint arXiv:2311.04926* (2023).
[99] Joy Idialu, Deborah Etsenake, and Norhan Abbas. [n. d.]. Whodunnit: Human or AI? ([n. d.]). https://plg.uwaterloo.ca/~migod/846/current/projects/07-Norhan-Deborah-Joy-report.pdf
[100] Maurice Isserman. 2003. Plagiarism: A Lie of the Mind. *The Chronicle Review* 49 (2003). Issue 34. http://chronicle.com
[101] Brandon Jaipersaud, Paul Zhang, Jimmy Ba, Andrew Petersen, Lisa Zhang, and Michael R. Zhang. 2023. Decomposed Prompting to Answer Questions on a Course Discussion Board. In *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky*, Ning Wang, Genaro Rebolledo-Mendez, Vania Dimitrova, Noboru Matsuda, and Olga C. Santos (Eds.). Springer Nature Switzerland, Cham, 218–223.
[102] Maurice Jakesch, Advait Bhat, Daniel Buschek, Lior Zalmanson, and Mor Naaman. 2023. Co-Writing with Opinionated Language Models Affects Users' Views. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 111, 15 pages. https://doi.org/10.1145/3544548.3581196
[103] Sajed Jalil, Suzzana Rafi, Thomas D. LaToza, Kevin Moran, and Wing Lam. 2023. ChatGPT and Software Testing Education: Promises & Perils. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 4130–4137. https://doi.org/10.1109/ICSTW58534.2023.00078
[104] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* 55, 12 (2023), 248:1–248:38. https://doi.org/10.1145/3571730
[105] Deborah G Johnson and Mario Verdicchio. 2023. Ethical AI is not about AI. *Commun. ACM* 66, 2 (2023), 32–34.
[106] Karl O. Jones, Reid Juliet, and Bartlett Rebecca. 2008. Cyber Cheating in an Information Technology Age. *Digithum* 10 (Dec. 2008). https://raco.cat/index.php/Digithum/article/view/394993
[107] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stepha Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. 2023. ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education. *Learning and Individual Differences* 103 (2023), 102274. https://doi.org/10.1016/j.lindif.2023.102274
[108] 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 *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[109] Tyson Kendon, Leanne Wu, and John Aycock. 2023. AI-Generated Code Not Considered Harmful. In *Proceedings of the 25th Western Canadian Conference on Computing Education* (Vancouver, BC, Canada) *(WCCCE '23)*. Association for Computing Machinery, New York, NY, USA, Article 3, 7 pages. https://doi.org/10.1145/3593342.3593349

[110] Arshia Khan and Janna Madden. 2018. Active Learning: A New Assessment Model That Boost Confidence and Learning While Reducing Test Anxiety. *International Journal of Modern Education and Computer Science* 10, 12 (2018), 1.

[111] Natalie Kiesler. 2020. On Programming Competence and Its Classification. In *Proceedings of the 20th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '20)*. Association for Computing Machinery, New York, Article 1, 10 pages. https://doi.org/10.1145/3428029.3428030

[112] Natalie Kiesler. 2020. Towards a Competence Model for the Novice Programmer Using Bloom's Revised Taxonomy - An Empirical Approach. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 459–465. https://doi.org/10.1145/3341525.3387419

[113] Natalie Kiesler. 2022. *Kompetenzförderung in der Programmierausbildung durch Modellierung von Kompetenzen und Informativem Feedback*. Dissertation. Johann Wolfgang Goethe-Universität, Frankfurt am Main. Fachbereich Informatik und Mathematik.

[114] Natalie Kiesler. 2024. *Modeling Programming Competency: A Qualitative Analysis*. Springer, Cham. https://doi.org/10.1007/978-3-031-47148-3

[115] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. In *CoRR 2309.00029*. https://doi.org/10.48550/arXiv.2309.00029 arXiv:2309.00029 [cs.AI]

[116] Natalie Kiesler and Daniel Schiffner. 2022. On the Lack of Recognition of Software Artifacts and IT Infrastructure in Educational Technology Research. In *20. Fachtagung Bildungstechnologien (DELFI)*, Peter A. Henning, Michael Striewe, and Matthias Wölfel (Eds.). Gesellschaft für Informatik e.V., Bonn, 201–206. https://doi.org/10.18420/delfi2022-034

[117] Natalie Kiesler and Daniel Schiffner. 2023. Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments. In *CoRR abs/2308.08572*. https://doi.org/10.48550/arXiv.2308.08572 arXiv:2308.08572 [cs.SE]

[118] Natalie Kiesler and Daniel Schiffner. 2023. Why We Need Open Data in Computer Science Education Research. https://doi.org/10.1145/3587102.3588860. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education Vol. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3587102.3588860

[119] Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. SPOC: Search-based Pseudocode to Code. *Advances in Neural Information Processing Systems* 32 (2019).

[120] Celine Latulipe, N Bruce Long, and Carlos E Seminario. 2015. Structuring Flipped Classes With Lightweight Teams and Gamification. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 392–397.

[121] Sam Lau and Philip J. Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile"': How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools Such as ChatGPT and GitHub Copilot *(ICER '23)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3568813.3600138

[122] Changyoon Lee, Yeon Seonwoo, and Alice Oh. 2022. CS1QA: A Dataset for Assisting Code-based Question Answering in an Introductory Programming Course. *arXiv preprint arXiv:2210.14494* (2022).

[123] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 124–130. https://doi.org/10.1145/3587102.3588785

[124] 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. arXiv:2304.03938 [cs.CY]

[125] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, NY, NY, USA, 563–569. https://doi.org/10.1145/3545945.3569770

[126] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level Code Generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097.

[127] 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]

[128] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGCSE Bull.* 36, 4 (jun 2004), 119–150. https://doi.org/10.1145/1041624.1041673

[129] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. *arXiv preprint arXiv:2305.01210* (2023).

[130] Dastyni Loksa, Lauren Margulieux, Brett A. Becker, Michelle Craig, Paul Denny, Raymond Pettit, and James Prather. 2022. Metacognition and Self-Regulation in Programming Education: Theories and Exemplars of Use. *ACM Trans. Comput. Educ.* 22, 4, Article 39 (sep 2022), 31 pages. https://doi.org/10.1145/3487050

[131] Rose Luckin, Wayne Holmes, Griffiths Mark, and Laurie B. Forcier. 2016. Intelligence Unleashed: An Argument for AI in Education. (2016). https://discovery.ucl.ac.uk/id/eprint/1475756/

[132] Andrew Luxton-Reilly. 2016. Learning to Program is Easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. ACM, NY NY, USA, 284–289. https://doi.org/10.1145/2899415.2899432

[133] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. Association for Computing Machinery, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[134] Qianou Ma, Tongshuang Wu, and Kenneth Koedinger. 2023. Is AI the Better Programming Partner? Human-Human Pair Programming vs. Human-AI pAIr Programming. *arXiv preprint arXiv:2306.05153* (2023).

[135] Stephen MacNeil, Joanne Kim, Juho Leinonen, Paul Denny, Seth Bernstein, Brett A. Becker, Michel Wermelinger, Arto Hellas, Andrew Tran, Sami Sarsa, James Prather, and Viraj Kumar. 2023. The Implications of Large Language Models for CS Teachers and Students. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1255. https://doi.org/10.1145/3545947.3573358

[136] Stephen MacNeil, Celine Latulipe, Bruce Long, and Aman Yadav. 2016. Exploring Lightweight Teams in a Distributed Learning Environment. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 193–198.

[137] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 931–937. https://doi.org/10.1145/3545945.3569785

[138] Stephen MacNeil, Andrew Tran, Juho Leinonen, Paul Denny, Joanne Kim, Arto Hellas, Seth Bernstein, and Sami Sarsa. 2023. Automatically Generating CS Learning Materials with Large Language Models. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1176. https://doi.org/10.1145/3545947.3569630

[139] Jordan K. Matelsky, Felipe Parodi, Tony Liu, Richard D. Lange, and Konrad P. Kording. 2023. A Large Language Model-assisted Education Tool to Provide Feedback on Open-ended Responses. arXiv:2308.02439 [cs.CY]

[140] Brent Daniel Mittelstadt, Patrick Allo, Mariarosaria Taddeo, Sandra Wachter, and Luciano Floridi. 2016. The Ethics of Algorithms: Mapping the Debate. *Big Data & Society* 3, 2 (2016), 2053951716679679. https://doi.org/10.1177/2053951716679679

[141] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2023. In-IDE Generation-based Information Support with a Large Language Model. arXiv:2307.08177 [cs.SE]

[142] Beatrice Nolan. 2023. Here are the Schools and Colleges That Have Banned the Use of ChatGPT over Plagiarism and Misinformation Fears. https://www.businessinsider.com/chatgpt-schools-colleges-ban-plagiarism-misinformation-education-2023-1

[143] Committee on Publication Ethics. 2023. Authorship and AI tools. https://publicationethics.org/cope-position-statements/ai-author Accessed: 08 July 2023.

[144] Michael Sheinman Orenstrakh, Oscar Karnalim, Carlos Anibal Suarez, and Michael Liut. 2023. Detecting LLM-Generated Text in Computing Education: A Comparative Study for ChatGPT Cases. arXiv:2307.07411 [cs.CL]

[145] Maciej Pankiewicz and Ryan S. Baker. 2023. Large Language Models (GPT) for automating Feedback on Programming Assignments. arXiv:2307.00150 [cs.HC]

[146] Michael Quinn Patton. 2002. *Qualitative Research & Evaluation Methods*. Sage, Thousand Oaks.

[147] Randy Pausch, Tommy Burnette, AC Capeheart, Matthew Conway, Dennis Cosgrove, Rob DeLine, Jim Durbin, Rich Gossweiler, Shuichi Koga, and Jeff White. 1995. Alice: Rapid Prototyping System for Virtual Reality. *IEEE Computer Graphics and Applications* 15, 3 (1995), 8–11.

[148] Fynn Petersen-Frey, Marcus Soll, Louis Kobras, Melf Johannsen, Peter Kling, and Chris Biemann. 2022. Dataset of Student Solutions to Algorithm and Data Structure Programming Assignments. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. 956–962.

[149] Carrie Anne Philbin. 2023. Exploring the Potential of Artificial Intelligence Program Generators in Computer Programming Education for Students. *ACM Inroads* 14, 3 (aug 2023), 30–38. https://doi.org/10.1145/3610406

[150] 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:2302.04662 [cs.PL]

[151] Tung Phung, Victor-Alexandru Pădurean, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors. *International Journal of Management* 21, 2 (2023), 100790.

[152] Stephen R Piccolo, Paul Denny, Andrew Luxton-Reilly, Samuel Payne, and Perry G Ridge. 2023. Many Bioinformatics Programming Tasks Can be Automated With ChatGPT. *arXiv preprint arXiv:2303.13528* (2023).

[153] Russell A Poldrack, Thomas Lu, and Gašper Beguš. 2023. AI-assisted coding: Experiments with GPT-4. *arXiv preprint arXiv:2304.13187* (2023).

[154] Lori Pollock. 2019. A Collaborative Practicum Targeting Communication Skills for Computer Science Researchers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 845–851. https://doi.org/10.1145/3287324.3287454

[155] Leo Porter and Daniel Zingaro. 2023. *Learn AI-Assisted Python Programming With GitHub Copilot and ChatGPT*. Manning, Shelter Island, NY, USA. https://www.manning.com/books/learn-ai-assisted-python-programming

[156] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. ACM, NY, NY, USA, 2–13. https://doi.org/10.1145/3372782.3406263

[157] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michael E. Caspersen, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. Transformed by Transformers: Navigating the AI Coding Revolution for Computing Education: An ITiCSE Working Group Conducted by Humans. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 2* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 561–562. https://doi.org/10.1145/3587103.3594206

[158] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/3230977.3230981

[159] 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. *ACM Trans. Comput.-Hum. Interact.* (aug 2023). https://doi.org/10.1145/3617367 Just Accepted.

[160] Ben Puryear and Gina Sprint. 2022. Github Copilot in the Classroom: Learning to Code with AI Assistance. *J. Comput. Sci. Coll.* 38, 1 (nov 2022), 37–47.

[161] Victor-Alexandru Pădurean, Georgios Tzannetos, and Adish Singla. 2023. Neural Task Synthesis for Visual Programming. arXiv:2305.18342 [cs.LG]

[162] Junaid Qadir. 2023. Engineering Education in the Era of ChatGPT: Promise and Pitfalls of Generative AI for Education. In *2023 IEEE Global Engineering Education Conference (EDUCON)*. 1–9. https://doi.org/10.1109/EDUCON54358.2023.10125121

[163] Rajendra Raj, Mihaela Sabin, John Impagliazzo, David Bowers, Mats Daniels, Felienne Hermans, Natalie Kiesler, Amruth N. Kumar, Bonnie MacKellar, Renée McCauley, Syed Waqar Nabi, and Michael Oudshoorn. 2021. Professional Competencies in Computing Education: Pedagogies and Assessment. In *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education* (Virtual Event, Germany) *(ITiCSE-WGR '21)*. Association for Computing Machinery, New York, NY, USA, 133–161. https://doi.org/10.1145/3502870.3506570

[164] Parsa Rajabi, Parnian Taghipour, Diana Cukierman, and Tenzin Doleck. 2023. Exploring ChatGPT's Impact on Post-secondary Education: A Qualitative Study. In *Western Canadian Conference on Computing Education (WCCCE'23), May 04–05, 2023*. Simon Fraser University.

[165] Arun Raman and Viraj Kumar. 2022. Programming Pedagogy and Assessment in the Era of AI/ML: A Position Paper. In *Proceedings of the 15th Annual ACM India Compute Conference* (Jaipur, India) *(COMPUTE '22)*. Association for Computing Machinery, New York, NY, USA, 29–34. https://doi.org/10.1145/3561833.3561843

[166] Raghu Raman, Santanu Mandal, Payel Das, Tavleen Kaur, Sanjanasri JP, and Prema Nedungadi. 2023. University Students as Early Adopters of ChatGPT: Innovation Diffusion Study. https://doi.org/10.21203/rs.3.rs-2734142/v1.

[167] Brent Reeves, Sami Sarsa, James Prather, Paul Denny, Brett A. Becker, Arto Hellas, Bailey Kimmel, Garrett Powell, and Juho Leinonen. 2023. Evaluating the Performance of Code Generation Models for Solving Parsons Problems With Small Prompt Variations. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 299–305. https://doi.org/10.1145/3587102.3588805

[168] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52 (11 2009), 60–67. Issue 11. https://doi.org/10.1145/1592761.1592779

[169] Steven I. Ross, Michael Muller, Fernando Martinez, Stephanie Houde, and Justin D Weisz. 2023. A Case Study in Engineering a Conversational Programming Assistant's Persona. In *Joint Proceedings of the ACM IUI Workshops 2023, March 2023, Sydney, Australia*.

[170] Gery W. Ryan and H. Russell Bernard. 2003. Techniques to Identify Themes. *Field Methods* 15, 1 (2003), 85–109. https://doi.org/10.1177/1525822X02239569

[171] Dana Saito-Stehberger. 2022. Examples of Culturally Responsive Teaching in Computational Thinking Curriculum. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1055. https://doi.org/10.1145/3478432.3499247

[172] Pamela Samuelson. 2020. AI Authorship? *Commun. ACM* 63, 7 (jun 2020), 20–22. https://doi.org/10.1145/3401718

[173] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. 2023. Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants. *arXiv preprint arXiv:2208.09727* (2023).

[174] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. ACM, NY NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[175] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. *arXiv preprint arXiv:2306.10073* (2023).

[176] Jaromir Savelka, Arav Agarwal, Christopher Bogart, and Majd Sakr. 2023. Large Language Models (GPT) Struggle to Answer Multiple-choice Questions About Code. *arXiv preprint arXiv:2303.08033* (2023).

[177] Jaromir Savelka, Arav Agarwal, Christopher Bogart, Yifan Song, and Majd Sakr. 2023. Can Generative Pre-trained Transformers (GPT) Pass Assessments in Higher Education Programming Courses? *arXiv preprint arXiv:2303.09325* (2023).

[178] Carsten Schulte and Jens Bennedsen. 2006. What Do Teachers Teach in Introductory Programming?. In *Proceedings of the Second International Workshop on Computing Education Research* (Canterbury, United Kingdom) *(ICER '06)*. Association for Computing Machinery, New York, NY, USA, 17–28. https://doi.org/10.1145/1151588.1151593

[179] Mazyar Seraj, Eva-Sophie Katterfeldt, Kerstin Bub, Serge Autexier, and Rolf Drechsler. 2019. Scratch and Google Blockly: How Girls' Programming Skills and Attitudes Are Influenced. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*. ACM, New York. https://doi.org/10.1145/3364510.3364515

[180] Judy Sheard, Angela Carbone, Raymond Lister, Beth Simon, Errol Thompson, and Jacqueline L. Whalley. 2008. Going SOLO to Assess Novice Programmers. *SIGCSE Bull.* 40, 3 (jun 2008), 209–213. https://doi.org/10.1145/1597849.1384328

[181] Judy Sheard, Simon, Matthew Butler, Katrina Falkner, Michael Morgan, and Amali Weerasinghe. 2017. Strategies for Maintaining Academic Integrity in First-Year Computing Courses. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITiCSE '17)*. ACM, NY, NY, USA, 244–249. https://doi.org/10.1145/3059009.3059064

[182] Lee S. Shulman. 2005. Signature Pedagogies in the Professions. *Daedalus* 134, 3 (2005), 52–59. http://www.jstor.org/stable/20027998

[183] Simon, Judy Sheard, Michael Morgan, Andrew Petersen, Amber Settle, and Jane Sinclair. 2018. Informing Students about Academic Integrity in Programming. In *Proceedings of the 20th Australasian Computing Education Conference* (Brisbane, Queensland, Australia) *(ACE '18)*. Association for Computing Machinery, New York, NY, USA, 113–122. https://doi.org/10.1145/3160489.3160502

[184] Adish Singla. 2023. Evaluating ChatGPT and GPT-4 for Visual Programming. arXiv:2308.02522 [cs.LG]

[185] Sarin Sok and Kimkong Heng. 2023. ChatGPT for Education and Research: A Review of Benefits and Risks. https://doi.org/10.2139/ssrn.4378735

[186] Pragnya Sridhar, Aidan Doyle, Arav Agarwal, Christopher Bogart, Jaromir Savelka, and Majd Sakr. 2023. Harnessing LLMs in Curricular Design: Using GPT-4 to Support Authoring of Learning Objectives. *arXiv preprint arXiv:2306.17459* (2023).

[187] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is ChatGPT the Ultimate Programming Assistant - How far is it? *arXiv preprint arXiv:2304.11938* (2023).

[188] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI '22)*. ACM, NY NY, USA, 1–7.

[189] Tianjia Wang, Daniel Vargas Dıaz, Chris Brown, and Yan Chen. 2023. Exploring the Role of AI Assistants in Computer Science Education: Methods, Implications, and Instructor Perspectives. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*.

[190] Tianjia Wang, Daniel Vargas-Diaz, Chris Brown, and Yan Chen. 2023. Towards Adapting Computer Science Courses to AI Assistants' Capabilities. *arXiv preprint arXiv:2306.03289* (2023).

[191] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 172–178. https://doi.org/10.1145/3545945.3569830

[192] Patricia Widjojo and Christoph Treude. 2023. Addressing Compiler Errors: Stack Overflow or Large Language Models? arXiv:2307.10793 [cs.SE]

[193] Alistair Willis, Patricia Charlton, and Tony Hirst. 2020. Developing Students' Written Communication Skills with Jupyter Notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1089–1095. https://doi.org/10.1145/3328778.3366927

[194] Claes Wohlin, Marcos Kalinowski, Katia Romero Felizardo, and Emilia Mendes. 2022. Successful Combination of Database Search and Snowballing for Identification of Primary Studies in Systematic Literature Studies. *Information and Software Technology* 147 (2022), 106908.

[195] Lixiang Yan, Lele Sha, Linxuan Zhao, Yuheng Li, Roberto Martinez-Maldonado, Guanliang Chen, Xinyu Li, Yueqiao Jin, and Dragan Gašević. 2023. Practical and Ethical Challenges of Large Language Models in Education: A Systematic Literature Review. arXiv:2303.13379 [cs.CL]

[196] Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. 2022. Wordcraft: Story Writing With Large Language Models. In *27th International Conference on Intelligent User Interfaces* (Helsinki, Finland) *(IUI '22)*. Association for Computing Machinery, New York, NY, USA, 841–852. https://doi.org/10.1145/3490099.3511105

[197] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large Language Models Meet NL2Code: A Survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 7443–7464.

[198] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in Computing Education: Perspectives of Students and Instructors. *IEEE Frontiers in Education (FIE)* (2023). https://doi.org/10.48550/arXiv.2308.04309

[199] Jialu Zhang, José Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. 2022. Repairing Bugs in Python Assignments Using Large Language Models. *arXiv preprint arXiv:2209.14876* (2022).

A PAPER EXTRACTION FORM

The following questions were placed on a form that the group used when evaluating the papers that were included in the literature review.

(1) **Paper title** [text entry]
(2) **Bibtex entry** [text entry]
(3) **Article type** [multi-select]
- Position / discussion paper
- Supervised study (a study that is conducted in a highly controlled environment like a research lab)
- Unsupervised study (a study that is conducted in a less restricted environment, such as online)
- New tool paper (presenting a new tool / LLM)
- Evaluation paper (evaluating an existing tool / LLM)
- Other [text entry]

(4) **Author affiliation** [multi-select]
- Academic
- Industry

(5) **Country of human participants**: If data is collected from human participants, provide a comma separated list of countries of where participants were located (if not explicitly mentioned, put "unclear"). [text entry]
(6) **Level of human participants** [multi-select]
- Uncontextualized
- Primary school (e.g. elementary, intermediate, middle school)
- Secondary school (e.g. high school)
- Tertiary education (e.g. college, university)
- Informal education (e.g. MOOCs)
- Professional developers
- Not applicable
- Other [text entry]

(7) **Number of human participants from whom data was collected** (if available, type into other) [multi-select]
- Not applicable
- Unclear
- Other [text entry]

(8) **Description of participants**: A copy-paste (or paraphrased) description of participants from whom data is collected which may be useful to a more detailed thematic analysis. This information can often be found at the beginning of a Methods section. [text entry]
(9) **How do the authors motivate the work**: A copy-paste (or paraphrased) description of the motivation for the work as expressed by the authors. [text entry]
(10) What LLM / tool is used? [multi-select]
- GPT-3
- GPT-4
- Codex
- Copilot
- Unclear
- N/A
- Other [text entry]

(11) **What are the explicit research questions / research goals / hypotheses in the article?** A copy-paste (or paraphrased) description of the RQs, goals, hypotheses. [text entry]
(12) **What programming languages are involved in the study?** [multi-select]
- Java
- Python
- C
- C++
- Not programming language focused
- Other [text entry]

(13) **How does the article evaluate the data collected?** [multi-select]
- Qualitatively
- Quantitatively
- N/A

(14) **Quality assessment**: An assessment of the research "quality". [Yes / No / Vague radio grid]
- **Is there a clearly defined research question/hypothesis?**
- **Is the research process clearly described?**
- **Are the results presented with sufficient detail?**
- **Are threats to validity / limitations addressed in an explicit (sub)section?** (code as "vague" if discussed, but not in a separate subsection)

(15) **What is the contribution / what are the key results of the article?** Provide a short summary of the main findings. [text entry]
(16) **Curriculum changes: Provide any potential effects on computing curriculum that could result from this work** (when answering this question, please feel free to provide some of your own commentary - it is fine to mention curriculum changes which the paper prompted you to think about, even if they aren't explicitly mentioned by the paper authors). [text entry]
(17) **Opportunities: Provide any potential opportunities for computing education that could result from this work** (when answering this question, please feel free to provide some of your own commentary - it is fine to mention opportunities which the paper prompted you to think about, even if they aren't explicitly mentioned by the paper authors). [text entry]
(18) **Threats: Provide any potential threats for computing education that could result from this work** (when answering this question, please feel free to provide some of your own commentary - it is fine to mention threats which the paper prompted you to think about, even if they aren't explicitly mentioned by the paper authors). [text entry]
(19) **Additional notes**: Can be used to note any interesting aspects of paper or anything else relevant that isn't captured in the extraction fields above. [text entry]

B STUDENT SURVEY QUESTIONS

The following questions were used in the survey filled out by students.

(1) **Gender**
- ○ Man
- ○ Woman
- ○ Non-binary
- ○ Other:

(2) **Country** (drop-down list)

(3) **Level of Study**
- ○ First Year
- ○ Second Year
- ○ Third Year
- ○ Fourth Year
- ○ Fifth Year
- ○ Other:

(4) **Degree Major / Specialization**
- ○ Computer Science
- ○ Software Engineering
- ○ Information Technology
- ○ Computer Engineering
- ○ Bioinformatics
- ○ Other:

(5) **Number of courses with a programming component which you have completed** (text entry)

(6) **Rate your agreement with the following statements:** (Likert)
- I regularly use GenAI tools when working with text (e.g.: writing emails, reports, summaries)
- I regularly use GenAI tools when working with code (e.g.: generating code or explanations, writing programs, debugging, ...)
- I regularly use GenAI tools when working with images (e.g.: generating new pictures, ...)

(7) **After generating code using GenAI tools, I mostly:**
- ○ Not applicable (I have not used GenAI tools to generate code)
- ○ Use the code immediately.
- ○ Skim through the code briefly to make sure that it looks correct.
- ○ Read it carefully (with scepticism) to ensure that it is correct.
- ○ Read it carefully (with scepticism) and also write code to test it.

(8) **Rate your agreement with the following statements:** (Likert)
- I expect to use GenAI tools increasingly in my learning practices in the future
- Using GenAI tools frequently to generate code is harmful for my learning of programming
- GenAI tools can provide guidance for coursework as effectively as human teachers
- GenAI tools will replace human teachers in the future

(9) **Students must be taught how to use GenAI tools well for their future careers** (Likert)

(10) **When you have a question regarding the material you are studying or are stuck on a problem, in what order do you do the following?** (ranking question)
- Ask using GenAI tools
- Ask on the course discussion forum
- Search online (e.g. Google)
- Ask a friend
- Ask on online forums such as Stack Overflow
- Ask the course instructor/TA

(11) **Rate your agreement with the following statements:** (Likert)
- The policies at my university are clear regarding what is allowed and what is not allowed in terms of using GenAI tools
- The policies in the courses I took last semester were clear regarding what is allowed and what is not allowed in terms of using GenAI tools
- There should be no restrictions on the use of GenAI tools in coursework

(12) **For programming assignments, I believe GenAI should be:**
- ○ Always allowed
- ○ Allowed in some assignments, disallowed in others (based on the assignment type, course level, etc.)
- ○ Always disallowed

(13) **Can you elaborate on when you believe GenAI should be allowed or disallowed?** (text entry)

(14) **To what extent do you think students at your school are using GenAI tools in ways that your instructors would not approve of?**
- ○ Almost everyone
- ○ Many
- ○ Some
- ○ A few
- ○ Almost none

(15) **In the absence of an explicit course policy on the use of GenAI tools, which of the following do you consider as NOT ethical? (Mark all that apply):**
- □ It is unethical to auto-generate a solution for the whole assignment (or a large portion of it) and submitting it without understanding it.
- □ It is unethical to auto-generate a solution for the whole assignment (or a large portion of it) and submitting it after reading it and completely understanding it.
- □ It is unethical to auto-generate a solution even for small parts of the assignment.
- □ It is unethical to use GenAI tools to "explain" to you (step-by-step) how to solve the problem.
- □ It is unethical to provide your code to GenAI tools and ask them to help you fix a bug.
- □ It is unethical to ask GenAI tools to comment, tidy and improve the style of your code.
- □ It is unethical to write the solution in a programming language (other than the one used in the course) and asking GenAI tools to translate it for you to the language of the course (and then submitting the translated code).

(16) **If everyone in class is using GenAI tools, but it is against the rules to use them, then I would still use them.** (Likert)
(17) **Rate your agreement with the following statements:** (Likert)
- GenAI tools will negatively impact my future job prospects
- GenAI tools will harm the development of generic or transferable skills such as teamwork, problem-solving, and leadership
- I am concerned that I will become over reliant on GenAI tools
- I trust the code written by GenAI tools more than the code I write
- My instructors can detect code that was written by GenAI tools
- My instructors actively check for unauthorized use of GenAI tools
- Even if my instructors disallow GenAI tools in my programming assignments, it is fine for me to use them to generate code as long as I understand the code very well. It is unethical only if I copy without understanding.

(18) **Describe the ways you currently use GenAI tools in computing courses for text generation (e.g.: writing reports, summaries, etc.)** (text entry)
(19) **Describe the ways you currently use GenAI tools in computing courses for code generation (e.g.: debugging, writing, etc.)** (text entry)
(20) **Describe the effects you think GenAI tools will have on your prospects for future employment:** (text entry)
(21) **What are your views on the allowed usage of GenAI tools in coursework/exams?** (text entry)

C INSTRUCTOR SURVEY QUESTIONS

The following questions were used in the survey filled out by instructors.

(1) **Gender**
- ○ Man
- ○ Woman
- ○ Non-binary
- ○ Other:

(2) **Country** (drop-down list)

(3) **Teaching Experience (years)** (text entry)

(4) **Select the sizes of classes you taught in the most recent semester**
- ☐ 1-10 students
- ☐ 11-30 students
- ☐ 31-50 students
- ☐ 51-100 students
- ☐ 101-250 students
- ☐ 251-500 students
- ☐ 500+ students

(5) **Select the type of department/school/faculty**
- ○ Computer Science
- ○ Software Engineering
- ○ Information Technology
- ○ Computer Engineering
- ○ Bioinformatics
- ○ Other

(6) **Rate your agreement with the following statements:** (Likert)
- I regularly use GenAI tools when working with text (e.g.: writing emails, reports, summaries)
- I regularly use GenAI tools when working with code (e.g.: generating code or explanations, writing programs, debugging, ...)
- I regularly use GenAI tools when working with images (e.g.: generating new pictures, ...)

(7) **After generating code using GenAI tools, I mostly:**
- ○ Not applicable (I have not used GenAI tools to generate code)
- ○ Use the code immediately.
- ○ Skim through the code briefly to make sure that it looks correct.
- ○ Read it carefully (with scepticism) to ensure that it is correct.
- ○ Read it carefully (with scepticism) and also write code to test it.

(8) **Rate your agreement with the following statements:** (Likert)
- I expect to use GenAI tools increasingly in my teaching practices in the future
- Using GenAI tools frequently to generate code is harmful for my students' learning of programming
- GenAI tools can provide guidance for coursework as effectively as human teachers
- GenAI tools will replace human teachers in the future

(9) **Rate your agreement with the following statements**
- Students must be taught how to use GenAI tools well for their future careers
- I plan to change my assessment practices now that GenAI tools are commonly available
- I plan to change my curriculum now that GenAI tools are commonly available

(10) **Rate your agreement with the following statements** (Likert)
- The policies at my university are clear regarding what is allowed and what is not allowed in terms of using GenAI tools
- The policies in the courses I taught last semester were clear regarding what is allowed and what is not allowed in terms of using GenAI tools
- There should be no restrictions on the use of GenAI tools in coursework

(11) **For programming assignments, I believe GenAI should be:**
- ○ Always allowed
- ○ Allowed in some assignments, disallowed in others (based on the assignment type, course level, etc.)
- ○ Always disallowed

(12) **Can you elaborate on when you believe GenAI should be allowed or disallowed?** (text entry)

(13) **To what extent do you think students at your school are using GenAI tools in ways that you would not approve of?**
- ○ Almost everyone
- ○ Many
- ○ Some
- ○ A few
- ○ Almost none

(14) **In the absence of an explicit course policy on the use of GenAI tools, which of the following do you consider as NOT ethical for students to do? (Mark all that apply):**
- ☐ It is unethical to auto-generate a solution for the whole assignment (or a large portion of it) and submitting it without understanding it.
- ☐ It is unethical to auto-generate a solution for the whole assignment (or a large portion of it) and submitting it after reading it and completely understanding it.
- ☐ It is unethical to auto-generate a solution even for small parts of the assignment.
- ☐ It is unethical to use GenAI tools to "explain" how to solve the problem step-by-step.
- ☐ It is unethical to provide code to GenAI tools and ask them to help fix a bug.
- ☐ It is unethical to ask GenAI tools to comment, tidy and improve the style of the code.
- ☐ It is unethical to write the solution in a programming language (other than the one used in the course) and asking GenAI tools to translate it to the language of the course (and then submitting the translated code).

(15) **Rate your agreement with the following statements** (Likert)
- GenAI tools will negatively impact my students' future job prospects

- GenAI tools will harm the development of generic or transferable skills such as teamwork, problem-solving, and leadership
- I am concerned that my students will become over reliant on GenAI tools
- I trust the code written by GenAI tools more than the code I write
- I can detect code that was written by GenAI tools
- I actively check for unauthorized use of GenAI tools

(16) **Describe the ways you currently use GenAI tools in computing courses for text generation (e.g.: writing reports, summaries, etc.)** (text entry)

(17) **Describe the ways you currently use GenAI tools in computing courses for code generation (e.g.: debugging, writing, etc.)** (text entry)

(18) **Please describe any changes you have made, or plan to make, to your teaching approaches in courses you are teaching:** (text entry)

(19) **Please describe any changes you have made, or plan to make, to your assessment approaches in courses you are teaching:** (text entry)

(20) **If you have already implemented changes, describe how successful you think they were.** (text entry)

(21) **What new content/courses do you think should be taught/added to the curriculum?** (text entry)

(22) **Does your institutional academic integrity policy explicitly mention generative AI?**
- ○ Yes
- ○ No

(23) **Can you provide a link to your institution's policy on GenAI tools?** (text entry)

(24) **Do you have a policy explicitly stated in your syllabus about when GenAI tools should or could be used in your course?**
- ○ Yes
- ○ No

(25) **If you have a statement about acceptable use of generative AI in your course syllabus documents, and you are willing to share that statement, please paste it below:** (text entry)

(26) **Have you observed any students using these models? If so, how are they using them?** (text entry)

(27) **Are you willing to be interviewed regarding the use of GenAI tools in computing classes? If so, please provide a preferred contact email address.** (text entry)

D STUDENT GUIDE

Generative AI refers to a kind of artificial intelligence software that is capable of generating information in response to prompts. The software is trained on source data, and uses that training data as input to a sophisticated model that predicts the appropriate response to the prompt. It does not understand the prompts, but it produces a convincing simulation of understanding. Examples of generative AI systems that use text include ChatGPT and Bard, and generative AI models capable of generating images include Midjourney and DALL-E.

Generative AI tools can be used in ways that increase productivity and help you to learn. However, they may also be used in unproductive ways that provide answers without helping you to learn.

> **Policy on generative AI:**
> - You may use AI tools to help you learn during lab exercises and assignments.
> - You will NOT be permitted to use AI tools in secure assessments (i.e., the Test and Exam).

Examples of productive use

Generative AI tools are used in industry so you will be likely to use them regularly in your future work after graduation. Therefore, you should learn to use them appropriately to receive the most long-term benefit. As a student, effective uses of generative AI tools are centered on helping you understand course material, and may include asking generative AI to:

- Explain a given topic, or to provide an example of how programming constructs are used.
- Explain your program one line at a time.
- Produce an example that is similar to assignment questions.
- Explain the meaning of error messages.
- Generate code to complete tasks that you have already mastered from previous coursework.

Examples of inappropriate use

Some uses of generative AI do not typically help you learn, and such uses are likely to result in worse long-term outcomes (e.g., you will not be able to complete Test and Exam questions, or to continue following courses that expect a mastery of early programming content). Examples of these uses are:

- Using AI tools on official assessments where it has been forbidden.
- Asking generative AI to complete laboratory questions or assignments for you.
- Asking generative AI to debug code that has errors.
- Writing a code solution in a language you know and then asking an AI tool to translate that code into the language required for the assignment.

Risks of generative AI

There are many risks associated with the use of generative AI.

Accuracy If you are using generative AI tools for learning then you should always double-check the content. For example, if you are assigned to write a program that uses a specific algorithm, AI tools may generate a solution that arrives at the correct answer but does not use the required algorithm. If you use generative AI to assist in the creation of assessed content then you are responsible for the accuracy and correctness of the work that you submit.

Quality Content generated may be of poor quality, and generic in nature. Code may have security flaws and may contain bugs. It is important that you understand how any generated code works and you evaluate the quality of the content.

Learning Generative AI can be a powerful productivity tool for users who are already familiar with the topic of the generated content because they can evaluate and revise the content as appropriate. Tasks assigned by your teachers are designed to help you learn, and relying on AI tools to complete tasks denies you the opportunity to learn, and to receive accurate feedback on your learning.

Over-reliance Using AI tools to do your work for you may achieve the short-term goal of assignment completion, but consistent over-reliance on AI tools may prevent you from being prepared for later examinations, subsequent coursework, or future job opportunities.

Motivation You may experience lack of motivation for tasks that generative AI can complete. It is important to understand that you need to master simple tasks (which generative AI can complete) before you can solve more complex problems (which generative AI cannot complete). Stay motivated!

Impact on others

There are many consequences to inappropriate usage of AI tools. Some of these consequences may be unintended, and could potentially harm others. For example:

Other students You could expose other students to harm by preventing their learning or including content in a group assignment that violates academic integrity.

Faculty Violating academic integrity standards through the use of AI tools requires time and energy, and is emotionally draining to teachers and administrators, to enforce these standards.

Institutional Including code from AI tools that you do not understand could expose the university to loss of reputation or even financial harm through lawsuits.

Academic misconduct

Using generative AI in ways that are not permitted will be treated as academic misconduct. This will have serious consequences.