Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation

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

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

James Finnie-Ansley
The University of Auckland
Auckland, New Zealand
james.finnie-ansley@auckland.ac.nz

Andrew Luxton-Reilly
The University of Auckland
Auckland, New Zealand
a.luxton-reilly@auckland.ac.nz

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

Eddie Antonio Santos
University College Dublin
Dublin, Ireland
eddie.santos@ucdconnect.ie

Figure 1: An image generated by Midjourney with the prompt "robot writing computer code while student watches, computer screens, computer programming, computer code, realistic, highly detailed, cinematic –aspect 16:9"

ABSTRACT

The introductory programming sequence has been the focus of much research in computing education. The recent advent of several viable and freely-available AI-driven code generation tools present several immediate opportunities and challenges in this domain. In this position paper we argue that the community needs to act quickly in deciding what possible opportunities can and should be leveraged and how, while also working on overcoming otherwise mitigating the possible challenges. Assuming that the effectiveness and proliferation of these tools will continue to progress rapidly, without quick, deliberate, and concerted efforts, educators will lose advantage in helping shape what opportunities come to be, and what challenges will endure. With this paper we aim to seed this discussion within the computing education community.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; **Computer science education**; **CS1**; • **Computing methodologies** → **Artificial intelligence**.

SIGCSE 2023, March 15–18, 2023, Toronto, ON, Canada

ACM ISBN 978-1-4503-9431-4/23/03.
https://doi.org/10.1145/3545945.3569759

KEYWORDS

AI; AlphaCode; Amazon; artificial intelligence; code generation; CodeWhisperer; Codex; Copilot; CS1; CS2; GitHub; Google; GPT-3; introductory programming; large language model; LLM; machine learning; Midjourney; novice programmers; OpenAI; programming; Tabnine

ACM Reference Format:
Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), March 15–18, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3545945.3569759

1 INTRODUCTION

Recent months have seen the release of several AI models that represent step-changes in their respective domains. Text-to-image models such as OpenAI's DALL-E 2 [34] and Midjourney[1] (see Figure 1) are revolutionizing how images are created, with the latter being called "the greatest artistic tool ever built, or a harbinger of doom for entire creative industries"[2]. In July 2022, it was announced that DeepMind's AlphaFold predicted the structure of nearly all 200 million proteins known to science and is making them freely

[1] midjourney.com/home
[2] newatlas.com/computers/dall-e-2-ai-art

available [14]. Also in the last year, OpenAI and DeepMind – among others – have released groundbreaking models that generate computer code. At this point it seems that most of these tools will cost money to use professionally but often be free for educational use and to students[3]. It is already evident that some computing students are using AI code generation during the completion of course work.

The introductory programming sequence has been the focus of much research over several decades [4, 26] and the challenges of programming at the level required of a first-year computing student have been debated extensively. Although it has been described as easy [25] and that describing it as "hard" is debatable and may even be harmful [3], programming is perceived as being difficult by many. As students face many barriers in learning to program, automated help is likely attractive to many.

With the recent advent of several viable AI-driven code generation tools, 'writing' code that will suffice to pass traditional first-year (CS1 and even CS2) programming assignments and even exams seems to have become much easier [12, 13, 35].

What does an introductory computing course look like when we can assume that students will be able to easily auto-generate code solutions to their lab and assignment tasks by merely pasting problem descriptions into an AI-powered tool? Further, how will this affect the delivery of computing curricula in general? Our view is that these tools stand to change how programming is taught and learned – potentially significantly – in the near-term, and that they present multiple opportunities and challenges that warrant immediate discussion as we adapt to the use of these tools proliferating.

2 BACKGROUND AND CONTEXT

Although long-sought, viable AI-driven code generation has only become a reality for the general public since 2021. Three large-scale AI-powered code generation tools – OpenAI Codex, DeepMind AlphaCode, and Amazon CodeWhisperer – as well as many others, have recently come into the spotlight. The advantages such systems claim to provide include making "programming more productive and accessible" [23]. They also bring several challenges.

2.1 OpenAI Codex

In September 2021 the *New York Times* published an article titled "A.I. Can Now Write Its Own Computer Code. That's Good News for Humans"[4] describing OpenAI's Codex model. OpenAI is a non-profit "AI research and deployment company"[5] set up in 2015 with a $1 billion pledge from several tech leaders and investors[6]. Codex is a descendant of GPT-3 (third-generation Generative Pre-trained Transformer) – one of the most advanced natural language models available today; often called Large Language Models (LLMs). To demonstrate the kind of output that GPT-3 can generate from a simple prompt, we asked GPT-3 (text-davinci-002) the following (after the outline of this paper was determined): *Write a paragraph on the educational opportunities and challenges presented by automated code generation tools.* Its response was:

> There are both educational opportunities and challenges presented by automated code generation tools. On the one hand, these tools can help students learn to code more quickly and efficiently. On the other hand, they can also lead to a false sense of understanding and proficiency, as students may become reliant on the tools to do the heavy lifting for them. Additionally, automated code generation tools can sometimes produce code that is difficult to read and understand, making it more challenging for students to debug and troubleshoot their programs.

It is worth noting that this single paragraph from GPT-3 includes some of the opportunities and challenges (in relation to AI-generated code) that we address in this paper.

Codex is trained on more than 50 million GitHub repositories including the vast majority of GitHub Python code, totaling 159 GB. Files deemed as likely to be auto-generated, those with an average line count greater than 100, those with maximum line length greater than 1000, and those containing a small percentage of alphanumeric characters were filtered [7]. Codex can take English-language prompts and generate code in several languages including JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, and shell, but is "most capable" in Python[7]. It can also translate code between programming languages, explain (in English) the functionality of code provided as input, and return the time complexity of code it generates. It also has the ability to generate code that uses APIs, allowing it to, for example, send emails and access information in various databases. Codex is available via the OpenAI API[8] and also powers GitHub Copilot[9] which is billed as "Your AI pair programmer" – an intentional reference to pair programming, a well-known software engineering [2] and programming education approach [28]. Copilot is now available for free to verified students and teachers.[10]

The Codex model has been shown to perform well when solving programming tasks presented in plain English. The paper announcing Codex solved 29% of the problems in a new evaluation set developed by the Codex developers to measure functional correctness for synthesizing programs from Python docstrings. This performance increased to 70% when repeated sampling was employed [7].

The first evaluation of Codex on introductory programming problems was reported by Finnie-Ansley et al. [12], who compared its performance on summative exam questions to that of students in an introductory course, and found that it outperformed almost 80% of the students in the course. In addition, it comfortably solved many variants of the classic Rainfall Problem [37], including one novel variant that had never been published.

2.2 DeepMind AlphaCode

In February 2022, DeepMind[11] announced AlphaCode[12] which, like Codex, utilizes a transformer-based model that "writes computer programs at a competitive level"[13]. It is trained on over 715 GB of GitHub code including programs written in C++, C#, Go, Java,

[3] techcrunch.com/2022/05/24/copilot-githubs-ai-powered-coding-tool-will-become-generally-available-this-summer
[4] nytimes.com/2021/09/09/technology/codex-artificial-intelligence-coding.html
[5] openai.com/about
[6] cnbc.com/2021/01/08/openai-shows-off-dall-e-image-generator-after-gpt-3.html
[7] openai.com/blog/openai-codex
[8] beta.openai.com
[9] copilot.github.com
[10] github.com/pricing
[11] deepmind.com
[12] alphacode.deepmind.com
[13] deepmind.com/blog/competitive-programming-with-alphacode

JavaScript, Lua, PHP, Python, Ruby, Rust, Scala, and TypeScript [23]. All files larger than 1 MB or with lines longer than 1000 characters, and duplicates of the same file (ignoring whitespace) were filtered from training data. Unlike Codex, AlphaCode was fine-tuned on a curated set of publicly released competitive programming problems called CodeContests.[14] In introducing AlphaCode, Li et al. claim that a stripped-down version of AlphaCode, without the modifications described in their paper, performs similarly to Codex, "however, problems used in the Codex paper and similar work consist of mostly simple task descriptions with short solutions – far from the full complexity of real-world programming" [23].

AlphaCode ranked in the top 54% of over 5,000 programming competition participants from the Codeforces platform, solving new problems requiring a combination of critical thinking, logic, algorithms, coding, and natural language understanding [23]. Based on these results, the authors estimate that AlphaCode has a Codeforces[15] rating of 1238 which is within the top 28% of users that participated in a contest in the last 6 months [23]. Li et al. also showed that AlphaCode does not duplicate sections of code from the training dataset when producing solutions, instead relying heavily on natural language problem descriptions to create original solutions. AlphaCode is not currently available as an API or otherwise.

2.3 Amazon CodeWhisperer

Amazon CodeWhisperer was announced in June 2022 [1]. Unsurprisingly a Google Scholar search (July 27, 2022) returned only four results for *amazon codewhisperer*, none of which pertain to the tool itself. CodeWhisperer is billed as "the ML-Powered Coding Companion"[16] which "helps improve developer productivity by providing code recommendations based on developers' natural comments and prior code" [1]. Based on (for instance) a developer comment describing a task, CodeWhisperer attempts to determine which cloud services and public libraries are best for the task, generates code, and presents this as a recommendation to the developer within the IDE. Like Codex and AlphaCode, it is trained on public data. It is also claimed that accuracy is directly proportional to the size of the training data [1] – a finding similar to that of Codex [38]. CodeWhisperer is currently available for free, subject to a waitlist.[17]

2.4 Other AI code generation products

Although Codex, AlphaCode, and CodeWhisperer are the most publicized AI-driven code generation platforms, several others exist including Tabnine(.com), Code4Me(.me) and FauxPilot[18] (based on SalesForce CodeGen [31]). Most of these tools are commercial offerings aimed at professional software developers, as one of the oft-touted (although currently unproven) advantages of AI-driven code generation is increased development productivity.

3 POSITION

Our position is the following: *AI-generated code presents both opportunities and challenges for students and educators in introductory programming and related courses. The sudden viability and ease of access to these tools suggest educators may be caught unaware or unprepared for the significant impact on education practice resulting from AI-generated code. We therefore urgently need to review our educational practices in the light of these new technologies.*

We take it as given that these tools will continue to be readily available to students (as they are currently), that adoption will increase, and that the capabilities of the tools will improve. In the following sections we describe some of the opportunities and challenges presented by AI code generating tools in the context of university-level novices learning to program in the present time. We largely focus on opportunities and challenges that are already well-documented in the computing education literature, and discuss how AI-generated code is likely to affect the landscape of areas that are already well-studied. Where available, we include evidence and results from the literature although the literature on the effects of cutting-edge AI code generation tools is in its infancy. We intend this presentation of opportunities and challenges to form the basis for the inevitable discussions about the role of code generation tools in our education practices.

In this work we do not discuss wider societal (e.g., economic [7], political [18]) considerations presented by AI-generated code, nor those specific to advanced/professional programmers (e.g., [8, 30]). While important issues, our focus is on how AI-generated code is likely to impact students and educators in introductory programming (and related) classrooms in the near term.

While any new technology brings with it both positive and negative impacts, the hope is always that long-term net effects are positive. The developers of Codex specifically note that they do not "expect the impact of this class of technologies to be net-negative; rather, risks merit particular attention ... because they may be subtle or require deliberate effort to address, whereas [they] expect the benefits to be more obvious and "automatic" from the perspective of most users and affected stakeholders" [7].

The challenges and opportunities presented here are not exhaustive, but rather starting points for ongoing discussions that we hope lead to best educational practices involving code generation tools.

4 OPPORTUNITIES

Any new tool, when effective and widely-available, poses opportunities for learning. Handheld calculators have been ubiquitous in mathematics education since the 1980s. A study in (US) grades K-12 statistically analyzing 524 effects from 79 separate studies recommended the use of calculators in all mathematics classes from kindergarten (approx. age 5) and up, including in testing situations for grades 5 (approx. age 10) and up [15]. It remains to be seen if AI-powered code generation will follow a similar path.

Opportunities noted by the developers of Codex and AlphaCode include: "the potential to be useful in a range of ways" including "help onboard users to new codebases; reduce context switching for experienced coders; enable non-programmers to write specifications; have [such tools] draft implementations; and aid in education and exploration" [7]; and "the potential for a positive, transformative impact on society, with a wide range of applications including computer science education, developer tooling, and making programming more accessible" [23].

[14] github.com/deepmind/code_contests
[15] codeforces.com
[16] aws.amazon.com/codewhisperer
[17] pages.awscloud.com/codewhisperer-sign-up-form.html
[18] github.com/moyix/fauxpilot

In this section we offer a number of avenues where AI-generated code tools present clear opportunities. Although some opportunities bring related challenges, in this section we focus on their benefits.

4.1 Code Solutions for Learning

4.1.1 Exemplar solutions. Students learning to program are typically encouraged to practice writing code by completing many short problems that are often auto-graded [33, 40]. However, students are sometimes unable to complete these, and even when successful often seek exemplar solutions. Unfortunately, instructors do not always have the time to prepare and publish model solutions for all the programming exercises that students engage in (which may include test and exam questions). AI-generated solutions provide a low-cost way for students to generate exemplar solutions to check their work when practicing [12].

4.1.2 Variety of solutions. Code generation tools can also be used to help expose students to the variety of ways that a problem can be solved. There are many different approaches for solving most problems, although novices do not always appreciate this. Thompson et al. argue that providing appropriate variation in programming instruction is important, as it helps learners appreciate the efficiencies of and differences in writing code in a variety of ways [39]. Eckerdal and Thuné make a similar argument, drawing on variation theory to state that teachers should make available resources that highlight dimensions of variation in concepts being studied [11]. For most non-trivial problems, code generation tools produce a variety of correct solutions that are offered to the programmer for selection. This was illustrated by Finnie-Ansley et al. who observed a great deal of variation in the solutions that Codex generated when solving the classic Rainfall problem [12].

4.1.3 Code review of solutions. Current assessment approaches in introductory programming courses often focus on code correctness, rather than code quality or style. With the ability to generate syntactically correct solutions automatically, assessment can focus on the differences between multiple correct solutions, and making judgments on the style and quality of solutions.

Extensive literature on peer review, including code reviews [16, 24] outline the many benefits from looking at a variety of solutions to a given problem. These benefits are reportedly present even when the code is flawed – there are benefits from looking at good solutions as well as poor ones. Code generation models could be used to generate solutions of varying (or unknown) quality, and these could be used for assessment tasks focusing on the evaluation of code quality to engage students at the higher levels of Bloom's taxonomy. This may prove useful for generating discussions around alternative approaches and the quality of solutions, and provide the basis for refactoring exercises [12].

Current models are effective at generating correct code, but to our knowledge, no studies have looked at the style of AI-generated code. We believe that future models will have more sophisticated methods of selecting high-quality code that adheres to style conventions. The developers of AlphaCode note that automatic code generation could make programming more accessible and help educate new programmers [23]. These models could suggest alternative, and more efficient or idiomatic ways of implementing programs, which could help learners to improve their coding style.

4.2 Producing Learning Resources

Generating high-quality learning resources is very time consuming and typically requires a high level of expertise. The potential for generating novel learning resources, like programming exercises, explanations of code, and worked examples at essentially an unlimited scale is an exciting avenue for future work.

4.2.1 Exercise generation. Very recent work by Sarsa et al. has shown that the Codex model is capable of producing novel learning resources from a single priming example [35]. They explored the generation of two types of resources – programming exercises and code explanations – finding that most of the generated exercises were both sensible and novel, and included an appropriate sample solution [35]. They also reported that the Codex model was very effective at producing contextualized problem statements, targeting certain thematic topics that were simply specified as input to the model as part of the priming example.

4.2.2 Code explanations. High quality explanations of code are a useful type of resource for helping learners develop a robust understanding of programming concepts. One of the widely publicized features of Codex is that it can generate explanations of complicated pieces code. The example of this functionality provided as part of the OpenAI playground uses the prompt: "Here's what the above class is doing: 1." (with the number at the end prompting the model to produce an enumerated list when describing a code fragment). In their study of learning resource generation, Sarsa et al. found that most of the explanations generated by Codex were thorough and correct [35]. Recent work by MacNeil et al. also explored different kinds of prompts and the diverse code explanations they lead to when using the GPT-3 large language model [27].

4.2.3 Illustrative examples. Texts and other learning resources typically provide examples that are used to learn the relationship between a described programming problem and a solution. These can be used to illustrate a given programming construct, algorithmic pattern, data structure, or mapping from problem to solution. Students use these examples as models that help them learn, and frequently express a desire for more examples than are available. Code generation tools provide a means of satisfying this desire and providing as many examples as needed. As AI code generation tools improve, this could lead to worked examples that include reasoning for coding decisions, similar to those generated by Minerva for mathematics problems [22]. Such examples are believed to lower cognitive load and result in more effective learning [19].

4.3 New Pedagogical Approaches

Teaching in CS1 typically focuses initially on syntax and basic programming principles, and it usually takes time for students to master these fundamentals. If code generation models can be used to solve the low level implementation tasks, this may allow students to focus on higher level algorithms. In a way, this is similar to the use of block-based environments that remove the complexities of syntax and allow students to focus on algorithmic issues. Teaching could initially focus more on algorithms and problem solving, relying on

automatic code generation for implementation, and delay detailed and nuanced discussions of syntax until later.

4.3.1 Explaining algorithmic concepts clearly. The way that prompts entered as input into code generation models are constructed affects their performance. In fact, "prompt engineering" has, since the introduction of LLMs, become a distinct (and nascent) endeavor in working with these models. Simplifying the problem description was found to significantly increase the success rate of AlphaCode. On a sample of difficult problems, simplifying the description to make the required algorithm more explicit increased the percentage of correct samples from 12% to 55% [23]. It was also found that sensitivity to consistent variable naming decreases with model size – random changes to variable names in problem descriptions mattered less. That is, models are increasingly able to capture relevant relationships between variables described in the problem formulation. Students can focus more on how to communicate algorithmic problems clearly, thereby providing a better description to code generation models that can then generate working solutions.

4.3.2 Alleviating programmer's writer's block. Anecdotally, students sometimes struggle with programmer's writer's block – that is, they don't know how to get started, or to continue once stopped. Vaithalingam et al. [41] found that Copilot helped students to get started with programming assignments by producing starter code, thereby offering the opportunity to *extend* code rather than struggling with a blank page. Such an approach may require us to shift focus towards rewriting, refactoring, and debugging code; however, this provides the opportunity to help students maintain forward momentum in an authentic environment where the need for evaluating, rewriting, and extending code is perhaps more important than writing every line of code from scratch [29].

4.3.3 Overcoming traditional barriers. Novices face many barriers in learning to program [4]. For instance, programming (compiler) error messages are a known barrier to student progress [3, 17] with poor readability being only one of many factors [10]. Recent work has demonstrated that Codex is capable of explaining error messages in natural language - often effectively - and that that it can also provide correct fixes based on input code and error messages [21]. It is likely that the efficacy of these approaches will improve in time, and that other barriers to novice learning may be similarly mitigated by using these tools effectively.

5 CHALLENGES

The availability AI-based code generation raises concerns that it could be used in ways that limit learning, or in ways that make the work of educators more difficult. The developers of Codex note that their tool "raises significant safety challenges, does not always produce code that is aligned with user intent, and has the potential to be misused" [7]. Similarly, the developers of AlphaCode note that "like most technologies, these models might enable applications with societal harms which we need to guard against, and desire to have a positive impact is not itself a mitigation against harm" [23]. In this section we present a number of challenges presented by AI-generated code tools. Although some of these may also present opportunities, in this section we focus on their challenges.

5.1 Ethical Issues

Academic integrity in computing is a complex issue, particularly when software development encourages reuse of code and collaborative practices [36]. The use of auto-generated code raises significant issues with respect to academic integrity and code reuse.

5.1.1 Academic misconduct. Prior work has shown that AI-based code generators can achieve better than average marks on actual student exams [13], can perform well on both standard programming questions such as Rainfall [12], and reliably generate correct code for common algorithms such as insertion sort and tree traversal [8]. We can assume that AI-generated code tools will be capable of completing assignments given to students learning programming.

Simon et al. [36] note that contract cheating is growing in prevalence and increasingly difficult to detect. This suggests that there is student desire to outsource graded work to others. Traditional outsourced solutions have risks that communication between student and provider may be breached, or that the solution may be shared (or reused) by the provider resulting in duplicate submissions that can be detected. AI-generated solutions vary [12], and do not require communicating with another person, producing similar results to contracted outsourcing for students with fewer inherent risks. This provides a low-risk/high-reward avenue for students focused on short-term grades rather than developing a deep understanding of content. This may exacerbate existing issues with detection of academic misconduct.

5.1.2 Attribution. Simon et al. [36] surveyed academics about the use of attribution for code obtained from outside sources, finding a diverse range of views on the acceptability of code reuse. This academic integrity quagmire becomes more complex with relatively opaque differences between standard code completion tools present in IDEs and plugins such as Copilot that will provide code suggestions that are indistinguishable from IDE code completion.

In other contexts, humans regularly (and acceptably) use spell-checkers, grammar-checking tools, and predictive text and email auto-reply suggestions – all machine-generated and often relying on AI – in daily life. In a programming context, most development environments support code completion that suggests machine-generated code. Distinguishing different forms of machine suggestions may be challenging for academics, and it is unclear if we can reasonably expect introductory programming students who are unfamiliar with tool support to distinguish different forms of machine-generated code suggestions. If students are unable to discern the differences in these, it would be arguably unjust to treat the use of AI-generated code as academic misconduct – at least when using an IDE that provides different forms of assistance. This raises a key philosophical issue: how much content can be AI-generated while still attributing the intellectual ownership to a human? This calls into question the very concept of plagiarism [9] and how we should interpret plagiarism and intellectual contribution with machine-supported generation of content.

5.1.3 Code reuse and licensing. There are also potential licensing issues that arise when new content is produced using code generation models, even when the model data is publicly-available [23]. Many different licenses apply to much publicly-available code and typically these licenses require authors to credit the code they used,

even when the code is open-source. When the use of that code comes about via an AI model, developers may end up using code that requires license compliance and not be aware that it does.[19] This is clearly an issue that extends beyond educational use of software, but as educators it is our role to inform students of their professional responsibilities when reusing code.

5.1.4 Sustainability. The sustainability of our education practices, and in particular the impact on our environment, is an ethical issue that we must acknowledge. Training AI models can consume significant energy. Brown et al. [6] report that GPT-3/Codex required more than several thousand petaflop/s-days of computation during the pre-training process. Further, due to size these models are hosted centrally and accessed remotely. At present these tools are likely not as efficient in terms of compute power and network traffic than more established web services, and their environmental costs are a sustainability concern that should be known to users.

5.2 Bias and Bad Habits

The issue of bias in AI is well known [5]. In addition to general bias (subtle or overt) that applies to almost all AI-generated outputs such as only representing certain groups of people, genders, etc., there are likely biases specific to AI code generation.

5.2.1 Appropriateness for beginners. Given that most of the code that these models are trained on is public, it is reasonable to question if the public code used for training is appropriate for novice students. For example, professionals (and the over-confident) are likely more amenable to posting their code publicly. This can be used to support an argument that, despite myriad examples that public code is often not very good, it is nonetheless on average of higher quality than non-public code. At least most public code is complete and subject to public scrutiny – something that can not be said for private code. In addition to this quality bias, the code styles of public code is likely different and possibly more advanced than that of a typical "blank slate" novice. Additionally, these styles and approaches may not match the desires of a student's instructor.

5.2.2 Harmful biases. The developers of Codex note that code generation models raise bias and representation issues beyond problematic natural language – notably that Codex can generate code with structure that reflects stereotypes about gender, race, emotion, class, the structure of names, and other characteristics [7]. Additionally, Codex "can be prompted in ways that generate racist, denigratory, and otherwise harmful outputs as code comments" [7].

5.2.3 Security. Although largely ignored for much of the short history of computing education, the requirement for novice programmers to begin learning secure coding practices has been well-documented in recent years [20]. Given this, the security of AI-generated code is extremely important, even in educational settings. It has been shown that such code can be insecure [32], and human oversight is required for the safe use of AI code generation systems [7]. CodeWhisperer claims to tackle security head-on by providing the ability to run scans on code to detect vulnerabilities [1] although this is currently untested. Chen et al. note future code generation models may be able to be trained to produce more secure code than the average developer, this is far from certain [7].

[19] githubcopilotinvestigation.com

5.3 Over-reliance

A key risk of code generation models in practice is user over-reliance [7]. Especially with tools like Copilot that embed support in IDEs, novices may quickly become accustomed to auto-suggested solutions leading to students not reading problem statements carefully – or at all – and not thinking about the computational steps needed to solve a problem.

5.3.1 Reinforcing behaviors that impede learning. An analysis of solutions generated by AlphaCode revealed that 11% of Python solutions were syntactically-incorrect and 35% of C++ solutions did not compile [23]. It is not known what the average compilation rate of submitted solutions for average introductory programming students studying these languages are, however it is clear that students using such tools would be dealing with code that has a high probability of being incorrect in some way. The Codex developers noted that it can recommend syntactically-incorrect code including variables, functions, and attributes that are undefined or outside the scope of the codebase and that "Codex may suggest solutions that superficially appear correct but do not actually perform the task the user intended" [7]. If suggested code is incorrect, students may lose trust in the feedback provided by IDEs, including error messages, warnings and other auto-generated forms of feedback.

6 CONCLUSIONS

AI-generated code is on the way to being firmly part of the programming education landscape, but we do not yet know how to adapt our practices to overcome the challenges and leverage the benefits. It seems obvious that software development in the future will feature an increasing amount of auto-generated code and the use of such tools by those training for programming roles and jobs, such as our students. At a minimum we believe this suggests a shift in emphasis towards code reading and evaluating rather than code generation – a pedagogical approach consistent with the theory of instruction advanced by Xie et al. [43]. Beyond pedagogy, it also demands that we examine the ethical implications of the use of these tools and that we guide our students through such ethical reflection. In a 2022 ITiCSE keynote, Titus Winters, Principal Software Engineer at Google, suggested it is at least as important to be an ethically-aware person as it is be a good programmer [42]. We believe AI-generated code coupled with demands from industry will force us to face ethical issues in computing education from the very beginning of the curriculum. Without quick, concerted efforts, educators will lose advantage in helping shape what opportunities come to be, and what challenges will endure – in a landscape that is changing faster than ever.

REFERENCES

[1] Desai Ankur and Deo Atul. 2022. Introducing Amazon CodeWhisperer, the ML-powered Coding Companion. https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/
[2] Kent Beck. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, Boston, Massachusetts.
[3] Brett A. Becker, Paul Denny, Raymond Pettit, et al. 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

[4] 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)*. ACM, NY, NY, USA, 338–344. https://doi.org/10.1145/3287324.3287432

[5] Emily M Bender. 2019. A Typology of Ethical Risks in Language Technology With an Eye Towards Where Transparent Documentation can Help. Presented at the *Future of Artificial Intelligence: Language, Ethics, Technology Workshop*.

[6] Tom Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language Models are Few-shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, et al. 2021. Evaluating Large Language Models Trained on Code. https://arxiv.org/abs/2107.03374.

[8] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, et al. 2022. GitHub Copilot AI Pair Programmer: Asset or Liability? https://doi.org/10.48550/arXiv.2206.15331

[9] Nassim Dehouche. 2021. Plagiarism in the Age of Massive Generative Pre-trained Transformers (GPT-3). *Ethics in Science and Environmental Politics* 21 (2021), 17–23.

[10] Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. ACM, NY, NY, USA, 480–486. https://doi.org/10.1145/3341525.3387384

[11] Anna Eckerdal and Michael Thuné. 2005. Novice Java Programmers' Conceptions of "Object" and "Class", and Variation Theory. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05)*. ACM, NY NY, USA, 89–93. https://doi.org/10.1145/1067445.1067473

[12] James Finnie-Ansley, Paul Denny, Brett A. Becker, et al. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Australasian Computing Education Conference* (NY NY, USA) *(ACE '22)*. ACM, Online, 10–19. https://doi.org/10.1145/3511861.3511863

[13] 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 *Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE '23)*. ACM, NY, NY, USA, 8 pages. https://doi.org/10.1145/3576123.3576134

[14] Melissa Heikkilä. 2022. DeepMind has Predicted the Structure of Almost Every Protein Known to Science. https://www.technologyreview.com/2022/07/28/1056510/deepmind-predicted-the-structure-of-almost-every-protein-known-to-science/

[15] Ray Hembree and Donald J Dessart. 1986. Effects of Hand-held Calculators in Precollege Mathematics Education: A Meta-analysis. *Journal for Research in Mathematics Education* 17, 2 (1986), 83–99.

[16] Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Peer Code Review in Higher Education. *ACM Trans. Comput. Educ.* 20, 3, Article 22 (Sept. 2020), 25 pages. https://doi.org/10.1145/3403935

[17] Ioannis Karvelas, Annie Li, and Brett A. Becker. 2020. The Effects of Compilation Mechanisms and Error Message Presentation on Novice Programmer Behavior. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. ACM, NY, NY, USA, 759–765. https://doi.org/10.1145/3328778.3366882

[18] Heidy Khlaaf, Pamela Mishkin, Joshua Achiam, et al. 2022. A Hazard Analysis Framework for Code Synthesis Large Language Models. https://doi.org/10.48550/arxiv.2207.14157

[19] Edwin M. Knorr. 2019. Reforming a Database Course to Address Cognitive Load by Using Worked Examples. In *Proceedings of the Western Canadian Conference on Computing Education* (Calgary, AB, Canada) *(WCCCE '19)*. ACM, NY NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3314994.3325083

[20] Jessica Lam, Elias Fang, Majed Almansoori, et al. 2022. Identifying Gaps in the Secure Programming Knowledge and Skills of Students. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. ACM, NY NY, USA, 703–709. https://doi.org/10.1145/3478431.3499391

[21] Juho Leinonen, Arto Hellas, Sami Sarsa, et al. 2022. Using Large Language Models to Enhance Programming Error Messages. https://doi.org/10.48550/arxiv.2210.11630

[22] Aitor Lewkowycz, Anders Andreassen, David Dohan, et al. 2022. Solving Quantitative Reasoning Problems with Language Models. https://doi.org/10.48550/arxiv.2206.14858

[23] Yujia Li, David Choi, Junyoung Chung, Julian Schrittwieser, et al. 2022. Competition-level Code Generation With AlphaCode. *Science* 378, 6624 (2022), 1092–1097. https://doi.org/10.1126/science.abq1158 arXiv:https://www.science.org/doi/pdf/10.1126/science.abq1158

[24] Andrew Luxton-Reilly. 2009. A Systematic Review of Tools That Support Peer Assessment. *Computer Science Education* 19, 4 (2009), 209–232.

[25] 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

[26] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, et al. 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)*. ACM, NY, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[27] Stephen MacNeil, Andrew Tran, Dan Mogil, et al. 2022. Generating Diverse Code Explanations Using the GPT-3 Large Language Model. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. ACM, NY NY, USA, 37–39. https://doi.org/10.1145/3501709.3544280

[28] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. 2002. The Effects of Pair-Programming on Performance in an Introductory Programming Course. *SIGCSE Bull.* 34, 1 (Feb. 2002), 38–42. https://doi.org/10.1145/563517.563353

[29] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer—An Investigation of How Developers Spend Their Time. In *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, IEEE, Florence, Italy, 25–35.

[30] Ekaterina A Moroz, Vladimir O Grizkevich, and Igor M Novozhilov. 2022. The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement. In *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*. IEEE, IEEE, St. Petersburg, Russia, 386–390.

[31] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, et al. 2022. A Conversational Paradigm for Program Synthesis. https://doi.org/10.48550/arxiv.2203.13474

[32] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 754–768. https://doi.org/10.1109/SP46214.2022.9833571

[33] Raymond Pettit and James Prather. 2017. Automated Assessment Tools: Too Many Cooks, Not Enough Collaboration. *Journal of Computing Sciences in Colleges* 32, 4 (2017), 113–121.

[34] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, et al. 2022. Hierarchical Text-conditional Image Generation With Clip Latents. https://doi.org/10.48550/arxiv.2204.06125

[35] 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

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

[37] E. Soloway. 1986. Learning to Program = Learning to Construct Mechanisms and Explanations. *Commun. ACM* 29, 9 (Sept. 1986), 850–858. https://doi.org/10.1145/6592.6594

[38] Alex Tamkin, Miles Brundage, Jack Clark, and Deep Ganguli. 2021. Understanding the Capabilities, Limitations, and Societal Impact of Large Language Models. https://doi.org/10.48550/arxiv.2102.02503

[39] Errol Thompson, Jacqueline Whalley, Raymond Lister, and Beth Simon. 2006. Code Classification as a Learning and Assessment Exercise for Novice Programmers. In *19th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2006)*. National Advisory Committee on Computing Qualifications, Wellington, New Zealand, 291–298.

[40] Dwayne Towell and Brent Reeves. 2010. From Walls to Steps: Using Online Automatic Homework Checking Tools to Improve Learning in Introductory Programming Courses. *ACET Journal of Computer Education and Research* 6, 1 (2010), 8 pages.

[41] 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*. ACM, NY NY, USA, 1–7.

[42] Titus Winters. 2022. The Gap Between Industry and CS Education. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. ACM, NY NY, USA, 2–3.

[43] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, et al. 2019. A Theory of Instruction for Introductory Programming Skills. *Computer Science Education* 29, 2-3 (2019), 205–253. https://doi.org/10.1080/08993408.2019.1565235 arXiv:https://doi.org/10.1080/08993408.2019.1565235