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

Sam Lau
UC San Diego
La Jolla, California, USA
lau@ucsd.edu

Philip J. Guo
UC San Diego
La Jolla, California, USA
pg@ucsd.edu

ABSTRACT

Over the past year (2022–2023), recently-released AI tools such as ChatGPT and GitHub Copilot have gained significant attention from computing educators. Both researchers and practitioners have discovered that these tools can generate correct solutions to a variety of introductory programming assignments and accurately explain the contents of code. Given their current capabilities and likely advances in the coming years, how do university instructors plan to adapt their courses to ensure that students still learn well? To gather a diverse sample of perspectives, we interviewed 20 introductory programming instructors (9 women + 11 men) across 9 countries (Australia, Botswana, Canada, Chile, China, Rwanda, Spain, Switzerland, United States) spanning all 6 populated continents. To our knowledge, this is the first empirical study to gather instructor perspectives about how they plan to adapt to these AI coding tools that more students will likely have access to in the future. We found that, in the short-term, many planned to take immediate measures to discourage AI-assisted cheating. Then opinions diverged about how to work with AI coding tools longer-term, with one side wanting to ban them and continue teaching programming fundamentals, and the other side wanting to integrate them into courses to prepare students for future jobs. Our study findings capture a rare snapshot in time in early 2023 as computing instructors are just starting to form opinions about this fast-growing phenomenon but have not yet converged to any consensus about best practices. Using these findings as inspiration, we synthesized a diverse set of open research questions regarding how to develop, deploy, and evaluate AI coding tools for computing education.

CCS CONCEPTS

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

KEYWORDS

AI coding tools, LLM, ChatGPT, Copilot, instructor perspectives

This work is licensed under a Creative Commons Attribution International 4.0 License.

ICER '23 V1, August 07–11, 2023, Chicago, IL, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9976-0/23/08.
https://doi.org/10.1145/3568813.3600138

ACM Reference Format:
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. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1 (ICER '23 V1), August 07–11, 2023, Chicago, IL, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3568813.3600138

1 INTRODUCTION

Although research areas such as neural networks, program synthesis [43], and natural language programming [75] have been under development for decades, over the past year a series of commercial product launches brought those ideas into wider public consciousness. For instance, in June 2022 the GitHub Copilot AI code generation tool [35] launched after a year in private beta. In late November 2022, the ChatGPT AI chatbot launched [82]. Some analysts estimated that it reached 100 million users in two months, the fastest growth of any app on record [47]. Then, less than three months later, both Microsoft and Google announced ChatGPT-like conversational AI integration into their web search engines [74, 91].

This recent growth in popularity of AI tools has raised widespread concerns about issues such as bias [21, 63], ethics [17], misinformation [57], data licensing and privacy [22], energy usage and climate impacts [100], and centralization of corporate power [114]. One specific concern on the forefront of many educators' minds is the fact that these tools can effectively solve homework assignments and exam problems across a wide variety of school subjects [50].

Within computing education in particular, researchers have discovered that AI tools can be especially effective for programming due to them being trained on billions of lines of open-source code [61] and due to code having a more constrained logical structure than free-form natural language. These tools can generate solutions to programming assignments and exam questions [32, 38, 39, 111] and explain the contents of code [33, 68, 97]. Given these current realities and extrapolating to a future where AI capabilities are likely to improve, **how are computing educators planning to adapt their courses in response to the growing proliferation of AI code generation and explanation tools?**

To gather a diverse set of perspectives on this question, we interviewed 20 introductory programming course instructors (9 women + 11 men) in universities across 9 countries (Australia, Botswana, Canada, Chile, China, Rwanda, Spain, Switzerland, United States) spanning all 6 populated continents. Our participants ranged from

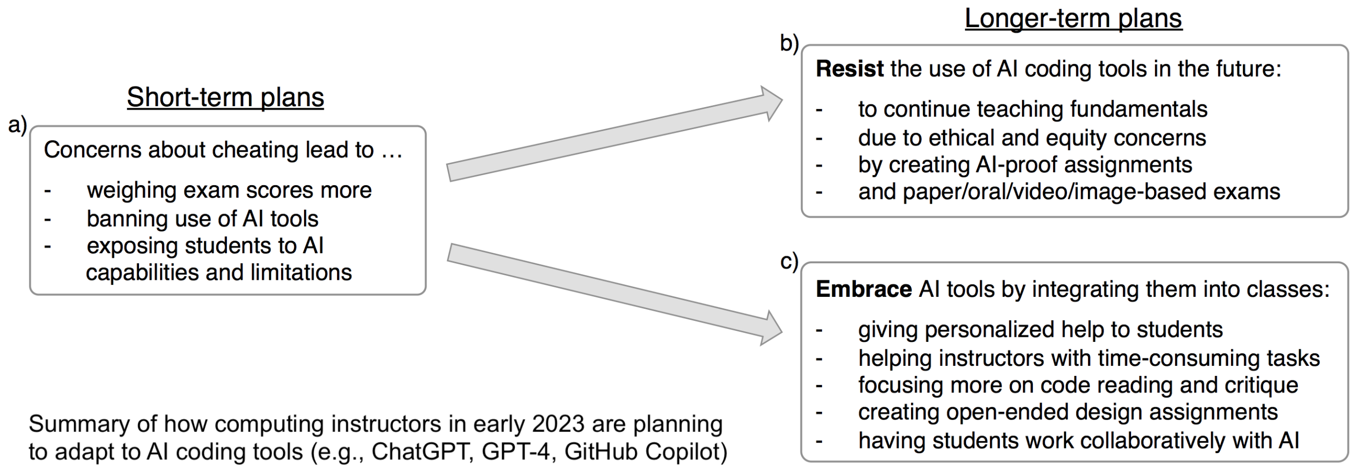
Short-term plans
a)
Concerns about cheating lead to …
- weighing exam scores more
- banning use of AI tools
- exposing students to AI capabilities and limitations
Longer-term plans
b)
Resist the use of AI coding tools in the future:
- to continue teaching fundamentals
- due to ethical and equity concerns
- by creating AI-proof assignments
- and paper/oral/video/image-based exams
c)
Embrace AI tools by integrating them into classes:
- giving personalized help to students
- helping instructors with time-consuming tasks
- focusing more on code reading and critique
- creating open-ended design assignments
- having students work collaboratively with AI
Summary of how computing instructors in early 2023 are planning to adapt to AI coding tools (e.g., ChatGPT, GPT-4, GitHub Copilot)

Figure 1: Summary of our study findings. We interviewed 20 introductory programming instructors and present both a) their short-term plans, and their longer-term plans to either b) resist or c) embrace the use of AI coding tools in their classes.

having zero prior experience with AI coding tools to having used them for personal programming projects.

Figure 1 summarizes our findings: a) In the short-term, all participants were concerned about cheating, which led to immediate reactions such as weighing exam scores more, banning the use of AI, or showing students the capabilities and limitations of AI tools. Longer-term, opinions diverged into two groups, with some wanting to b) resist the use of AI tools and continue teaching programming fundamentals, while others wanted to c) embrace AI tools by integrating them into their classes to help both students and instructors. Participants brainstormed a range of ideas for both resisting and embracing AI in future classes, ranging from creating 'AI-proof' assignments that may deter AI tools to new kinds of assignments where students must collaborate with AI.

Over the past academic year (2022–2023) computing education researchers have been actively discussing these topics in blog posts [19, 54, 55], a SIGCSE position paper [14], and workshops [65, 67]. Our paper complements these ongoing discussions by presenting *the first empirical study of computing instructor perspectives on AI code generation and explanation tools.* The timing of our study is unique since our interviews occurred in early 2023, which is the first academic term where large numbers of students have access to these tools due to ChatGPT's release in late 2022. Thus, our findings capture a rare snapshot in time when computing instructors are recounting their early reactions to this fast-growing phenomenon but have not worked out any best practices yet.

Since we are still very early in the adoption curve of AI coding tools, we hope our study findings can spur conversations within the computing education community about whether to resist or embrace these tools in the coming years, and how to work with these tools in ethical and equitable ways. We have a unique and timely opportunity to develop both policies and social norms that influence how these tools may impact future generations of students. Thus, we conclude this paper with a set of open research questions derived from our study findings (Section 7).

In sum, the contributions of this paper are:

- A comprehensive snapshot of the current state of AI coding tools and the range of human-centered research surrounding them as of early 2023, less than a year after the public release of ChatGPT and GitHub Copilot.
- The first study of computing instructors' perceptions of AI coding tools, which found that they were most concerned about cheating in the short term but that longer-term their sentiments bifurcated into either wanting to resist these tools or to embrace them by integrating them into future classes.
- A set of open research questions for the computing education community to consider as AI coding tools potentially grow more widespread in the coming years.

2 BACKGROUND: THE CURRENT STATE OF AI CODING TOOLS IN EARLY 2023

Over the past year (2022–2023), AI code generation and explanation tools have become more widespread with the release of products like GitHub Copilot (currently free for students and instructors) and ChatGPT (currently free for the public). These are built upon neural network models trained on terabytes of textual data scraped from the public internet (e.g., billions of webpages, billions of lines of open-source code from GitHub, and the contents of open-licensed books [24, 85]). Their large-scale architecture enables them to 'learn' patterns from data and generate text that a human might plausibly write, which is why they are commonly called Large Language Models (LLMs) [24]. And since code is a structured form of text, these tools can also synthesize code; thus, some refer to AI code generation as "Large Language Model (LLM)-driven program synthesis" [11, 55]. For brevity, throughout this paper we use the terms 'AI coding tools' or simply 'AI tools' as shorthand to refer to these tools. Users interact with these AI tools in three main ways:

1) Standalone: The simplest interface to LLMs is a web application that shows a text box, such as the OpenAI Playground [84] for GPT-series LLMs [20, 24, 85] (e.g., GPT-4) and TextSynth for various open-source LLMs [36]. The user can input some text (called a 'prompt' [112]) and the tool tries to generate text (or code) that plausibly continues from the user's input. An example code-generation prompt might be *"Write a Python function that adds two numbers."*

2) Conversational: Chatbots such as ChatGPT [82] improve upon a standalone interface by enabling users to hold a back-and-forth multi-turn conversation with the AI. This allows users to refer back to prior context instead of needing to re-enter their entire prompt every time. For instance, a user could say *"Now rewrite this code using more descriptive variable names"* and ChatGPT knows that the user is referring to code that was written earlier in the conversation.

3) IDE-integrated: Tools such as GitHub Copilot [35], Replit Ghostwriter [93], Amazon CodeWhisperer [8], Codeium [2], and Tabnine [110] integrate into the user's IDE (e.g., Visual Studio Code). This lets them do autocomplete and generate suggestions within the context of the user's codebase as they are coding. Another benefit of IDE integration is that these tools can pull in the user's own surrounding code (both before and after the cursor, plus in other open project files [94]) as context, which enables them to generate personalized code suggestions to fit the user's current task. Also, some IDE-integrated AI tools include an embedded chat interface.

As an indicator of just how fast things are moving in this space, many new LLMs and AI coding tools have been announced in the 2.5 months between when this paper was submitted (mid-March 2023) and when the final camera-ready publication was completed (early June 2023). Examples include new coding-capable LLMs such as LLaMA [103], GPT-4 [83], Cerebras-GPT [34], CodeGen2 [79], DIDACT [70], replit-code [5], and StarCoder [61]; LLM-based chatbots such as Alpaca [102], Claude [1], Dolly [29], Koala [42], and Vicuna [26]; and new IDE-integrated AI tools such as Sourcegraph Cody [7], Google's Codey LLM (similar name but unrelated tool!) integrated into Colab and Android Studio IDE [90], and Meta's CodeCompose [78]. GitHub also announced new Copilot X [3] enhancements, which include IDE-embedded AI chat interfaces.

2.1 Current Capabilities AI Coding Tools

To provide context for the early-2023 era when our study's interviews took place, we now summarize the current capabilities of AI coding tools that are most relevant for educational use cases.

2.1.1 *Code generation capabilities.*

Given natural language and/or code as input, these tools can generate relevant code:

Specification-to-code: Given a natural language description for what a piece of code should do (e.g., a function or class specification), these tools can generate code to meet that specification. For example: *"Create a function that takes a list of first names and a list of last names then returns a new list with those names joined."* Note that many CS1/CS2 programming assignments are phrased as specifications that students can directly input into tools.

Conversational specification-to-code: The main limitation of 'specification-to-code' is that novices are not good at writing precise specifications, so the generated code may not be what they want. To overcome this limitation, one can use the 'flipped interaction prompt pattern' [112] to have a back-and-forth conversation with ChatGPT before it generates the requested code. For instance, the user could write: *"ChatGPT, I want you to write a Python function to join first and last names. Ask me clarifying questions one at a time until you have enough information to write this code for me."*[1]

Code completion: Tools that integrate into the IDE, such as Copilot [35], Ghostwriter [93], CodeWhisperer [8], and Tabnine [110], enable the user to start typing code and see a list of contextually-relevant code completions, like an AI-enhanced autocomplete.

Code refactoring: Once the user has written some code, they can ask the tool to rewrite it in order to improve readability, style, or maintainability; e.g., *"Refactor this function to use smaller helper functions."* Again, holding a conversation with ChatGPT and answering its follow-up questions can help it to generate better code.

Code simplification: One kind of refactoring that students may try is to ask the tool to simplify a piece of code. For instance, *"Rewrite this code using only simple Python features that a student in an introductory programming course would know about."* Students could use this technique to generate more 'plausible-looking' answers to CS1/CS2 assignments, because otherwise it may look suspicious if they turn in code that uses too many advanced language features.

Language translation: These tools can also translate code written in one programming language into another (albeit imperfectly). They can also translate mentions of *human* languages within code.

Test generation: Users can also ask AI tools to generate test cases (e.g., unit/regression tests). For instance, Copilot has a TestPilot feature [72] that generates tests and interactively refines them based on user feedback. These tools can often create tests for unusual edge cases that novices may not think of on their own [113].

Structured test data generation: AI tools can also generate structured data that users can pass into their software to manually test it. For instance, one could ask a tool to generate 100 fake user profiles (with fake names, ages, and locations) in some format, like JSON or a Python dictionary, in order to test a prototype social media app.

2.1.2 *Code explanation capabilities.*

Given a piece of code and natural language instructions as input, these tools can explain what that code does in a way that emulates how a human instructor might explain it to a student:

Explanations at varying expertise levels: The most straightforward prompt is to ask the tool to explain what a piece of code does. One can also use the 'persona prompt pattern' [112] to generate explanations for a given expertise level. For instance, *"ChatGPT, I want you to take on the persona of a university CS1 instructor talking to a student who has never taken a programming class before. Explain what this function does: [user's code]."* Users can also ask it to automatically generate code comments or API documentation.

Debugging help: Users can ask the tool to find possible bugs in the given code and explain why those may be bugs. Note that the tool does not run the code or perform rigorous static code analysis. But in practice, 'superficial' bugs in student code (e.g., off-by-one errors in a loop bound) can be found by the tool matching against patterns learned from billions of lines of open-source code. And with ChatGPT, one can ask follow-up clarifying questions and engage in a back-and-forth debugging conversation, which simulate some aspects of working with a human tutor [71].

[1]The prompts in this paper are simplified illustrative examples that may not work optimally as-is. In practice, it likely takes a fair amount of iteration to craft prompts that work reliably and effectively. This process is known as *prompt engineering* [112].

Conversational bug finding: A more powerful way to do AI-assisted bug finding is to start a back-and-forth debugging conversation. For instance: *"Here is my code and the output I see when I run it. This output looks wrong because the last two array elements are duplicated. What should I change in my code to help me find the bug more easily?"* The AI tool might suggest a code edit; then the user can apply that edit, re-run their modified code, and paste the new text output or error message into the next round of dialogue. This conversational technique elegantly bypasses the AI tool's limitation that it cannot directly run the user's code – instead, the user runs the code on their computer and sends the output to the AI tool.

Code review and critique: AI tools can also serve as a code reviewer and give detailed critiques. Again, the 'persona pattern' [112] can be useful here, e.g.,: *"I want you to take on the persona of a senior software engineer at a top technology company. I have submitted this code to you for a formal code review. Please critique it: [user's code]"*

Conceptual explanations with code examples: The user can also ask the tool to explain a programming concept just like how they would ask a human instructor. For instance: *"What's the difference between checked and unchecked exceptions in Java? Give code examples for each."* The tool's ability to generate code examples can enable some basic level of (albeit imperfect) fact-checking since the user can run that code and see if it matches the given explanation.

2.2 Notable Limitations of AI Coding Tools

Here are some commonly-known limitations of these tools and other reasons people have cited for not using them:

- Inaccuracies: The most notable limitation is that these tools can generate inaccurate outputs with no quality guarantees [81]. This may result in subtly-inaccurate code or incorrect explanations which appear believable to novices.
- Code quality: They may generate code that is stylistically non-ideal, that may not be robust to edge cases, that have security vulnerabilities [86], or that is not aligned with what students are learning in a particular class.
- Knowledge cutoff: These tools only 'know' what is in their training data, which is an older snapshot of the web. So they cannot help with, say, a JavaScript library released last week. That said, they do get periodically re-trained, and Microsoft and Google are augmenting them to search the web [74, 91]. Also, tools like Sourcegraph Cody [7] augment an LLM by retrieving code and text from a user's own project repository in order to generate responses about facts that are not on the public web (a form of retrieval-augmented generation [60]).
- Learning curve: Novices may have a hard time producing high-quality results with simple prompts [115]. It takes some level of expertise to craft effective and reliable prompts [112].
- Nondeterminism: AI tools can produce different outputs even when given the same prompt. There are settings to reduce randomness of outputs, but whenever the underlying AI models get updated, results can still end up non-reproducible.
- Offensive content: AI tools can generate outputs that exhibit harmful biases [17, 63]. For instance, AI-generated code examples may contain offensive stereotypes embedded in variable names or strings [14, 18, 24].
- Ethical objections: Some people are opposed to using AI tools due to concerns about their creators disregarding software licenses when scraping code repositories for training data [22], the environmental impact of training and running LLMs [100], and companies using underpaid human workers to label and filter training data [89].

3 RELATED WORK

There are fast-growing lines of research on the technical architecture of LLMs (large language models) that power AI code generation tools[2], applications of LLMs to many specific domains (e.g., creative writing [77]), and broader societal implications of LLMs [17]. Instead of surveying the entire landscape of research on LLMs, we focus our discussion on parts of the literature that are the most relevant to our interview study. This encompasses a range of human-centered research on how LLM-based code generation tools relate to the fields of software engineering and computing education.

3.1 How Software Developers Use AI Code Generation Tools

Several recent groups of researchers have studied how software developers use GitHub Copilot in practice, since it is marketed as a tool to help developers be more productive [52]. Bird et al. combined forum analysis, a think-aloud study, and a survey to gather usage patterns such as Copilot enabling faster code-writing but at the expense of less code understanding [18]. Sarkar et al. analyzed blog and forum posts to give a similar overview [96]. Cheng et al. studied how developer communities might build trust in AI tools [25]. Barke et al. found that developers used it in two ways: to help them explore options and to accelerate their path toward a known goal [13]. Peng et al. found in a 95-user between-subjects study that using Copilot helped developers complete a web development task 56% faster than the control group [87]. But Vaithilingam et al. found in a 24-user within-subjects study that although developers liked using Copilot as a starting point, it did not always improve task completion time or accuracy [106].

More broadly, HCI researchers have done usability studies of AI code generation tools and proposed improved interface designs. For instance, Jayagopal/Lubin et al. analyzed the usability of 5 program synthesis tools (Copilot was one of them) and found that those which run in the background (without explicit user triggering) can be more learnable for novices [49]. Sun et al. discovered users' needs for explainability in AI code generation tools [101]. Vaithilingam et al. prototyped 19 user interface ideas for augmenting IDEs with AI assistance [105]. Ross et al. augmented an IDE with a conversational AI tool (similar to ChatGPT) called the Programmer's Assistant [95]. McNutt et al. proposed a design space for how to integrate AI assistance within computational notebooks, which have different affordances than IDEs [73]. Liu et al. proposed a user experience enhancement that translates the user's prompts into code and then back again to natural language in order to clarify what the tool intends to do [64].

[2]Some examples of LLMs specialized for programming include CodeBERT [37], PyMT5 [28], Codex [24], AlphaCode [62], CodeGen [80], CodeGen2 [79], Parsel [116], InCoder [40], CodeT [23], StarCoder [61], CodeCompose [78], replit-code [5], and Codey [90]. Note that modern general-purpose LLMs such as GPT-4 [83] are trained on large amounts of code as well, so they can also do code generation and explanation.

Although our study focuses on computing instructors and not software developers, some of the instructors we interviewed mentioned that their goal is to train the next generation of developers. Thus, instructors have been thinking about how to prepare students for a future where they might use these AI tools on the job.

3.2 Computing Education Research on AI Tools

When these AI tools first came out, computing education researchers were curious about whether they could solve programming assignments that are typically given in CS1 and CS2. For instance, Finnie-Ansley et al. found that Codex[3] could solve both CS1 [38] and CS2 [39] exam problems better than most students and also performed well on variations of the classic Rainfall Problem [98]. Wermelinger followed a similar study protocol but instead used Copilot within an IDE (instead of standalone Codex) to qualitatively understand the user experience of coding with an AI tool in an IDE [111]. Denny et al. then showed that using prompt engineering (i.e., adjusting the wording of the input prompt to Copilot) can significantly improve results when prompts are phrased more like step-by-step pseudocode [32]. Vahid et al. found that ChatGPT could solve all of the programming assignments in several CS1 courses either by directly copying the assignment prompt into ChatGPT or, if that did not work, by telling it which autograder test cases failed and having it correct itself [104]. A student copied all the 2022 'AP Computer Science A' free-response questions into ChatGPT, and it scored 32 out of 36 points [99].

A complementary line of research uses AI tools to assist instructors in creating course content: MacNeil et al. used these tools to generate explanations for CS1-level code [68] and then embedded them within an interactive e-book to see how students engage with them [66]. Sarsa et al. used Codex to automatically generate programming exercises and code explanations, both of which could be used to help students to get extra practice and guidance [97]. Denny et al. extend this idea by combining Codex with learnersourcing (i.e., crowdsourcing using learners) to generate and validate exercises in a way that is personally motivating to learners [33]. Lastly, Leinonen et al. use Codex to generate a specialized type of code explanation: enhanced compiler and run-time error messages [59].

Lastly, an emerging line of work measures the impact of AI tools on learners. Prather et al. observed how students used Copilot on CS1 assignments in a lab study and then interviewed them about their first impressions [92]. And Kazemitabaar et al. ran a controlled study with 69 pre-college students where half used Codex to learn Python and the other half did not [53]. They found that the Codex group could write code better while demonstrating a similar level of understanding as the control group.

These projects all focus on applying or extending AI coding tools. Our study complements their findings by reporting the perspectives of instructors regarding how they plan to prepare for a future where these tools become more widespread.

3.3 Perspectives of Computing Instructors

In terms of methodology, the closest related studies to ours are those that uncover the perspectives of computing instructors. For instance, Mirhosseini et al. interviewed 32 computing instructors across 5 countries to ask about the day-to-day challenges they face when running their courses [76]. Although their study did not ask about AI tools, they found that "having more examples or variety of assignments could benefit students both as additional resources as well as a way to prevent plagiarism." [76] AI tools can potentially help generate these more varied examples and assignments [67, 97]. Krause-Levy et al. interviewed 21 CS instructors to get their views on the purpose of prerequisite coursework [56]. Valstar et al. interviewed 14 faculty [107] then surveyed 249 faculty [108] to discover how they felt CS programs should be preparing students for industry jobs. In contrast to these prior studies, to our knowledge *our study is the first to interview computing instructors about their perspectives on AI code generation and explanation tools.*

Of recent note is a SIGCSE position paper that shares similar motivations as our study. The authors warn that "the sudden viability and ease of access to these [AI] 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." [14] Although that paper was not an empirical study, it poses relevant questions such as: "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?" Our interview protocol (Section 4) shares some similarities with this question, and some of our findings confirm what the authors foresaw as challenges and opportunities of AI tools (see Section 6 for details). Also, two other events at SIGCSE 2023 – a workshop on generating course materials using AI tools [67] and a Birds-of-a-Feather session on the implications of AI for computing instructors and students [65] – indicate ongoing community interest in the topics that our paper covers.

4 METHODS

To gather instructors' perspectives on AI tools, in early 2023 we conducted semi-structured interviews with 20 instructors who teach introductory programming courses at universities across 9 countries. Each was done by one researcher over Zoom videoconferencing, lasted 45 minutes to 1 hour, and was video-recorded upon getting verbal consent from the participant. Our interview protocol was semi-structured and began with three background questions:

- What is your level of personal experience with AI code generation and explanation tools?
- How much do you think that students are using these AI tools right now?
- How much have you heard your colleagues discussing these AI tools? (And in what settings?)

The purpose of these background questions is to establish a baseline for each participant's perceptions of the status quo in early 2023. They also help get the participant into the frame of mind to discuss our main open-ended question:

> Imagine a future where all students had an AI tool that can: 1) automatically write code to 'perfectly' solve any programming problem in your classes and be undetectable by plagiarism detectors since AI tools

[3] Codex is the LLM that GitHub Copilot was originally built on, although newer versions of Copilot may move to GPT-4 [3] since Codex was discontinued in March 2023.

Table 1: We interviewed 20 introductory programming instructors to elicit their thoughts on the use of AI code generation and explanation tools in their classes. The 'Years' column indicates years of experience as a full-time instructor so far.

ID	Gender	Age	Country	Years	Type of university	Prior experience with AI coding tools
P1	F	35-44	U.S.	9	Private PhD-granting	experimented with ChatGPT in her class
P2	F	35-44	U.S.	8	Public PhD-granting	no usage
P3	M	35-44	Chile	8	Private PhD-granting	minimal usage (ChatGPT)
P4	M	45-54	U.S.	16	Private PhD-granting	minimal usage (Copilot)
P5	M	65-74	Switzerland	35	Public PhD-granting	no usage
P6	F	55-64	U.S.	23	Private liberal arts	minimal usage (ChatGPT)
P7	M	35-44	U.S.	8	Public PhD-granting	used for programming and in his class
P8	F	35-44	Botswana	3	Public PhD-granting	no usage
P9	F	35-44	U.S.	14	Public PhD-granting	no usage, purposely avoiding AI tools for now
P10	M	45-54	Rwanda	15	Private PhD-granting	no usage
P11	M	55-64	U.S.	25	Private liberal arts	minimal usage (ChatGPT)
P12	F	35-44	U.S.	6	Private liberal arts	no usage
P13	M	25-34	China	2	Public PhD-granting	minimal usage (ChatGPT)
P14	M	45-54	Canada	23	Public undergrad-only	lots of usage for programming
P15	F	35-44	U.S.	2	Public PhD-granting	used for programming
P16	F	25-34	U.S.	4	Public PhD-granting	lots of experience using AI tools in her class
P17	F	45-54	U.S.	14	Private liberal arts	minimal usage (ChatGPT)
P18	M	45-54	Spain	20	Public PhD-granting	no usage
P19	M	55-64	Australia	33	Public PhD-granting	no usage, but has AI research experience
P20	M	25-34	U.S.	3	Public undergrad-only	experimented with ChatGPT, AI researcher

generate diverse code variants (not exact copies), and 2) explain what any piece of code does in English so that it can answer free-response homework questions for students too.

Walk me through your CS1/CS2 course materials and let's brainstorm how you would help students to learn effectively given this possible future. What might you do in both the short-term and longer-term?

We displayed this question on-screen via Zoom screen-share and spent the majority of each interview focused on it.

4.1 Rationale for Our Interview Protocol

The design of our interview protocol was guided by several theoretical considerations: First, we framed the interview session as a *speculative futures* brainstorming exercise [46] where we encouraged the participant to imagine a possible future where these AI tools are ubiquitous (*"Imagine a future where all students had an AI tool that can ..."*). This methodology comes from the field of speculative design [10], where the goal is to propose new designs without being bound by present-day constraints. In our context it means that we are *not* doing a usability study of ChatGPT, Copilot, or any specific AI tool; rather, we are assessing how instructors respond to hypothetical future tools in that vein. Aside from our main guiding question, we fostered a sense of idea exploration throughout the interview, reassuring participants that there are no 'right' or 'wrong' answers in the curriculum ideas they were proposing to us.

Next, we grounded each conversation in concrete artifacts that the participant presented to us via Zoom screen-share (*"Walk me through your CS1/CS2 course materials ..."*). This technique was inspired by the *cognitive walkthrough* [69] methodology in HCI, where having an artifact to discuss (such as course materials) can help more substantive ideas come out of brainstorming sessions. One risk here is that participants might get fixated on low-level details, so we also gave them time to do higher-level reflections before and after walking through their course materials.

To reduce cognitive biases such as priming [12] or anchoring [41] effects, we purposely did *not* mention specific tools such as ChatGPT or Copilot in our interview protocol. Everything was worded generically as 'AI tools.' However, if participants started talking about specific tools, then we let the conversation naturally turn to discussing the details of those tools.

4.2 Interview Participant Backgrounds

Table 1 summarizes the backgrounds of the 20 computing instructors we interviewed. We recruited participants from amongst our professional networks and using personal referrals from colleagues. We sought out instructors who taught programming-based university CS1 or CS2 courses since those are many students' entry points into computing. Also, AI tools may have more immediately-visible effects on CS1/CS2 since recent research has shown that they can already solve many kinds of CS1/CS2 programming assignments [32, 38, 39, 111].

Table 1 shows that we aimed for diversity across multiple dimensions such as gender (9 female + 11 male), age (ranging from 20s to 70s), years of full-time experience as an instructor (from 2 to 35 years, mean=13.5 years), type of university, and country. Most notably, our participants worked in universities across 9 countries (Australia, Botswana, Canada, Chile, China, Rwanda, Spain, Switzerland, United States) spanning all 6 populated continents.

4.3 Data Overview and Analysis

One researcher conducted each interview via Zoom and took timestamped notes during the interview. Then a second researcher independently watched each Zoom video recording and took their own set of notes. Both researchers met regularly to discuss their notes and watch excerpts of videos together. Throughout this process, we iteratively came up with a set of themes using an inductive approach [30]. We made several iterations together as a team before finalizing on our split into short-term and long-term curriculum ideas that participants mentioned in their interviews, with long-term itself then split into two groups (Section 5.3 and Section 5.4, respectively). We originally positioned sets of ideas as 'dimensions' along a continuous spectrum like a design space diagram [45, 58]. However, we realized that the ideas that participants proposed were more discrete in nature (e.g., using paper exams vs. computer-based exams) rather than falling along a spectrum, so we ended up grouping our findings using the format summarized by Figure 1.

4.4 Study Scope and Limitations

We scoped our study to CS1 and CS2 university instructors since, at this time, most prior research on AI coding tools focus on these introductory courses [32, 33, 38, 39, 68, 97, 111]. Thus, upper-division university courses, K-12 settings, and informal learning environments outside of schools are beyond the scope of our study, so our findings may not generalize to them.

Next, we designed our interview study as a speculative futures brainstorming exercise. Thus, if that future does not materialize (e.g., if AI tools stop being developed or are made less accessible) then the findings from our study may not be as applicable for future researchers or practitioners. Relatedly, while we asked participants to brainstorm how they might adapt their courses around a hypothetical future AI tool, in practice they often talked about a *specific present-day tool* such as ChatGPT or Copilot because those tools are what they have heard about or tried firsthand. Although we encouraged participants to generalize beyond current tools, some of our findings may still be tied to what participants think about the present-day capabilities of AI tools rather than what future tools might look like.

Lastly, even though our participants came from 9 countries across 6 continents, the majority were still from universities in the United States. Thus, despite our efforts to recruit globally, our findings are not as globally-representative as we would ideally like [16]. And since we conducted all the interviews in English, we are likely missing out on the diverse experiences of instructors worldwide who teach programming in other natural languages [44].

5 FINDINGS: COMPUTING INSTRUCTORS' PERCEPTIONS OF AI CODING TOOLS

Although our 20 interview participants teach courses in a wide range of institution types and locations (see Table 1), everyone felt that their learning environments would likely change in response to the growing presence of AI coding tools. To organize the themes that surfaced from our interviews, this section begins with what instructors have heard so far about these tools (Section 5.1) and any short-term changes they are now making to their courses (Section 5.2). Then we present the ways they envision their courses changing longer-term in response to AI, which fall into two possible futures: one where students are discouraged from using AI tools in introductory programming courses (Section 5.3), and another where these courses embrace AI tools (Section 5.4).

5.1 What do instructors currently know about AI coding tools in early 2023?

We began each interview with questions about each participant's personal experience with AI coding tools, how much they think students are now using them, and how much their colleagues are discussing this topic. Their responses represent a baseline as of early 2023, a few months after ChatGPT launched on Nov 30, 2022.

Personal experience so far: Although programming assistance tools have existed for many years (e.g., code autocomplete in IDEs), participants mentioned that AI-based tools only came to the forefront of their attention over the past year. A few were early adopters of trying GitHub Copilot for personal programming in 2022 (e.g., P7, P14, P15), but the majority started being aware of AI coding tools after the release of ChatGPT at the end of 2022. The rightmost column of Table 1 shows that roughly half tried out ChatGPT to varying degrees, ranging from casual personal use to experimenting with integrating it into their classes already. Eight participants reported never having used these tools yet ('no usage' in Table 1), but all had heard of them being discussed by others. P9 *purposely avoided* using AI tools for ethical reasons since she told her students not to use them so she wants to follow the same rules herself.

How much do they think students are using AI coding tools? Since for many instructors this is their first full teaching term where ChatGPT is available, nearly everyone we interviewed responded with some variant of "I don't know." The general sentiment was that as instructors it was hard for them to get a sense of whether students were using AI tools because students would likely *not* tell the professor about it. P7 asked one of his undergraduate TAs about whether she had witnessed students using it, and the TA responded that it almost felt like a taboo topic. The TA said she was afraid to be seen in the computer lab with ChatGPT open in her web browser (even for innocuous use cases) out of fear that she may be setting a bad example for her students; she said that it felt like being caught browsing a website that offered cheating services. P14 was the only one who saw direct evidence of students using AI tools. For some assignments he requires students to submit screenshots of their computer, and he has seen screenshots with the ChatGPT website open in a browser tab with the start of the assignment prompt clearly visible. He told us that *"they don't even try to hide it."*

How much are their colleagues discussing AI tools? However, despite not having a clear sense of how much students were using AI coding tools, *all* participants had heard their colleagues discussing these tools in recent months. These discussions ranged from informal hallway chats to faculty mailing list threads all the way to official department committees formed to investigate policies around AI tools. For instance, P6 said that over 20 faculty across her university contributed to a 57-message-long email thread reacting to the implications of AI tools for teaching. P9 said that her department formed a committee to investigate and eventually make policy decisions about AI usage in computing courses. P13 was the only one who reported their university issuing an official policy decision; he showed us a policy document banning the use of ChatGPT and other AI tools in all classes across the university.

5.2 Short-term concerns about cheating lead to immediate course adjustments

Although our interview questions encouraged participants to think longer-term about future uses of AI tools in their classes, everyone started their conversation by bringing up the topic of cheating as an immediate short-term concern that is on their minds. Note that all 20 participants brought up the topic of AI-assisted cheating near the start of their interviews *even though we did not mention cheating as a topic in our interview questions.*

A common concern was that students who relied on AI code generation tools to get the answers would not be learning the material. For example, a student who used an AI tool to complete an entire assignment could receive a good grade without meeting the learning objectives. P12 and P15 both independently pointed out that since future AI tools may become more seamlessly integrated into IDEs, some students might be *unintentionally* using them without realizing they are activated, thus hindering their learning.

Even though students already have access to outside resources such as Stack Overflow, assignment solutions leaked on the web, and peers who can help them, instructors were especially concerned about AI tools because these tools generate variations of code that are not exact copies and are thus less detectable by plagiarism detectors. Instructors with more prior AI experience, such as P14, said that one way to detect the use of these tools is if students submit assignments containing more advanced programming constructs (e.g., Python list comprehensions or lambdas) that have not yet been taught in an introductory course.

This concern was compounded by the pervasive belief among instructors that AI tools had inherent limitations – even though these tools could get "95%" of the problem correct, they would never be able to produce correct code in all scenarios, as P15 described: *"In the real world, programming is all about edge cases, so you get 95% there and that other 5% is the tricky part. Copilot is pretty good with the 95%."* Similarly, P15 noted how *"a lot of times the code it [GitHub Copilot] creates is fairly subtly incorrect because your situation is just slightly different from where it learned from, so you usually have to tweak one or two things."* Thus, she perceived that not only would students using AI tools be cheating, but they would be *"cheating badly"* since their code would be incorrect in subtle ways that they would not be able to understand.

Short-term adjustments to courses: In response to concerns about academic integrity, the majority of participants (14 out of 20) were already making adjustments now during the current term. Some common examples include:

- **Weigh exam scores more heavily**: P3, P5, P7, P12, P15, P17, and P18 adapted by weighing exam scores more in students' final grades, which involved no changes to course content. The rationale for doing so is that exams are taken under controlled environments so it is presumably harder to use AI tools to cheat on them. But P14 noted that exams are not cheat-proof: He witnessed a recent cheating case where a student had a smartphone hidden in their lap, took photos of their paper exam pages, sent those photos to friends in a WhatsApp group, and then those friends used ChatGPT to generate the solutions and message it back to them.
- **Ban AI tools in class**: P2, P4, P9, and P13 added bans of AI tools into their syllabuses this term. They showed us their syllabus and described how they wrote the bans using language that equated AI with other forms of code plagiarism. A few others, such as P6 and P17, referred students to their university's honor code when they had questions about AI, but they did not issue a course-wide ban since they wanted to deal with usage on a case-by-case basis. P17, P19, and P20 argued against such bans, with P19 remarking that *"it would be impossible to enforce, so why bother?"* Similarly, both P17 and P20 said that an official ban would only arouse students' curiosity since a ban could be interpreted as the school admitting that AI tools were effective.
- **Expose students to the capabilities and limitations of AI tools**: P1, P7, and P16 took the opposite approach by showing students what AI tools can and cannot do. P1 added an optional exercise where students use ChatGPT to solve a programming problem and turn in a chat transcript annotated with their reflections. P7 showed a live coding demo in class where he copied in homework questions from a prior term into ChatGPT and asked his class to critique the AI-generated code to assess its strengths and weaknesses. P16 did something similar by annotating how ChatGPT solved her class's prior homeworks. Then she let students use any AI tool they wanted on a take-home coding exam and reflect on their experiences. Note that these activities were easy to add to their current courses because they did not involve creating brand-new assignments. The rationale here was to show students that these tools were imperfect so if they wanted to use them then they had to carefully scrutinize the generated code, which may itself be a learning opportunity.

Participants remarked that these imperfect short-term patches were the best they could do right now given time constraints. The sudden appearance of ChatGPT at the end of 2022 meant that when they started teaching in January 2023, it was the first term when lots of students got access to a tool that could potentially solve their programming assignments. The last time they taught their current class, which was usually 6 to 12 months ago, AI tools were not nearly as widespread or easily accessible to students.

5.3 Longer-term ideas (1 of 2): Resisting AI tools may improve programming pedagogy

While everyone started their interviews by discussing cheating as a short-term concern (Section 5.2), eventually the conversation turned to brainstorming longer-term ideas that they could potentially implement over the next year or few years. These ideas were either about 1) resisting the use of AI tools in introductory programming courses (this section), or 2) embracing AI tools and integrating them into new curriculum (Section 5.4).

Why resist? Participants wanted to resist using AI tools in introductory programming courses due to:

- **Importance of learning programming fundamentals**: The most common reason given here is that participants felt it is still important to learn the fundamentals of programming, even if AI tools will be doing a lot of the coding in the future. Several made the oft-repeated analogy to math education after the introduction of calculators: Students still learn the fundamentals of arithmetic and algebra even though calculators can do all of those routine operations. P1 said that using AI coding tools is like *"giving kids a calculator and they can play around with a calculator, but if they don't know what a decimal point means, what do they really learn or do with it? They may not know how to plug in the right thing, or they don't know how to interpret the answer."* P1 and P5 both made an analogy to power tools versus hand tools for performing physical labor. P5 said that AI is like a power tool that professionals use but novices should start with the equivalent of hand tools (i.e., writing code themselves) to understand the fundamental principles before graduating to power tools. P1 said that *"it feels a bit weird to give them [CS1 students] this power so early."* P8 brought up the concept of fragile knowledge [88] and how she was concerned that if students use code that they did not write themselves, then the mental models they build about that code may not be robust; she likened it to how she currently sees students copying code from Stack Overflow into their projects and using that code without trying to understanding it first.

- **Ethical objections to AI**: P9 brought up the lawsuit filed against GitHub [22] for potentially violating the software licenses of open-source code repositories they used to train the Codex model that powers Copilot. She said that since the legal ramifications of these AI tools have not been clarified yet, it may be unethical for her to even teach her students how to use them: *"What if my students use AI at their job and their company gets sued, that's not good!"* She does not want to risk teaching her students to do something that may turn out to be illegal. Similarly, P17 brought up how these models ingest not only open-source code but terabytes of written and image content created by people who did not consent to have their work used in AI tools without attribution.
- **Potential lack of equity and access**: P6 raised questions about who is providing the data for training these AI systems, whether that data is representative of certain groups of people, and whether users may unknowingly reinforce biases by programming using AI tools trained on such data. P9 was concerned that students who are more 'in the know' about technology trends will learn how to use AI tools from peers while those without much prior technology exposure will not (i.e., an AI *digital divide* [31]). Thus, even though she is currently banning AI tools in her class, she may consider teaching with them in the future in order to share this knowledge with students in a more equitable way so that everyone starts off with access to this same baseline knowledge.

How to resist? Participants proposed the following ways to resist AI tools in their courses, operating under the assumption that more and more students will gain access to these tools in the future. They cannot stop students from using AI since even an official ban in the syllabus cannot always be enforced, so they want to redesign their curricula to mitigate its effects.

- **Designing AI-proof assignments**: One set of ideas for resisting AI tools involved designing assignments to be more 'AI-proof.' Several participants mentioned how traditional CS1/CS2 assignments consisting of self-contained programming tasks that are autograded with a test suite are no longer viable since AI can solve them [32, 104]. One way to improve upon them is by adding more *local context*. For instance, P7 walked us through a Java CS1 assignment that used thousands of lines of starter code to wrap around the Twitter API, which he and his TAs had written just for this class. He tried putting in his assignment questions into ChatGPT and, unsurprisingly, it could not generate good solutions since it lacked the context of his starter code. P4 and P6 showed us similar setups involving locally-written libraries of code, with P6 walking us through an assignment that uses a custom graphics library she developed for her class. These instructors felt that creating context-specific assignments with bespoke starter code may be a good way to stay ahead of AI tools' capabilities.[4] P3 (from Chile) and P8 (from Botswana) described a different form of local context: cultural and language context. They both wanted to incorporate local slang and cultural references from their home regions into programming assignments because their hunch is that AI tools trained on U.S./English-centric web data would likely not be knowledgeable about those nuances and would not be able to produce code to solve those assignments.
- **Bringing back paper exams**: P1, P4, P6, and P10 proposed going back to paper-based exams to assess learning since it would be harder to cheat using AI in this format. In recent years, especially during remote-only classes due to the global pandemic, there has been more of a trend toward computer-based exams in CS1/CS2, with the benefits being that students can run, test, and debug their code. However, students need to install special software that locks down their browser or records their session or even webcam, which can feel invasive [9]. Paper exams eliminate the need for such software and make it harder (but not impossible) for students to use devices to cheat since proctors can scan the room to make sure that only pencil and paper are present. P1 supported this idea but acknowledged that exams can feel too high-stakes and stress-inducing. So she proposed giving more frequent lower-stakes quizzes throughout the semester and letting students drop their 3 lowest quiz scores.
- **Oral, video, and image-based assessments**: Aside from paper exams, some participants also brainstormed other forms of assessment that could both 1) prevent the use of AI tools and 2) assess student learning in more meaningful ways. P7 and P10 brought up using oral exams where a TA would question students live one-on-one, though they acknowledged the challenges of scaling to large classes. To scale better, P7, P9, and P13 wanted students to video-record

[4]A few weeks after these interviews, Sourcegraph announced Cody [7], which uses retrieval-augmented generation [60] to take in the context of a user's entire custom codebase. Future AI tools will likely be able to ingest large bespoke codebases as well.

themselves tracing through code execution or explaining code that they just wrote. P3 brought up the fact that since AI tools can take only text instructions as input, he wanted to create assignments where the inputs are images, such as sketching out what he wants the student's code to do.[5]

- **Process-based assessments**: P9 mentioned that in software engineering courses, students are already graded in part based on the *process* of engineering (e.g., making feature branches and pull requests on GitHub, doing code reviews for teammates), not just on the final output. Can we adapt something similar for CS1/CS2? This may discourage students from using AI tools since they will not be able to explain their design process in depth. (But future AI tools might be able to come up with convincing process explanations too!)

Some participants predicted that their ideas for resisting AI tools *may actually improve pedagogy* in introductory courses. For instance, oral- and video-based assessments can get students to think more deeply about why code works the way that it does rather than simply writing code to obtain a known answer. And contextualized programming assignments can be more motivating to students than the generic questions that AI tools can now solve.

5.4 Longer-term ideas (2 of 2): Embracing AI tools for forward-looking CS curricula

The second set of longer-term ideas involved why and how to embrace AI coding tools in the coming years. Note that some of these ideas came from the same people who raised objections in the prior section, which indicates that participants were not universally pro- or anti-AI. Notably, several such as P2 who had put short-term bans on AI tools in their current classes were open to the idea of allowing AI in the future if they had time to adapt their curricula. P2 mentioned: *"I feel like the class would have to change a lot because almost all the questions we ask students could be solved by this thing [ChatGPT]. We're gonna have to change what we're asking of them."*

Why embrace? Participants gave four main reasons why they wanted to integrate AI tools into their curricula.

- **Preparing students for future jobs**: The most common reason instructors gave for integrating AI tools into CS1/CS2 was that they felt responsible for preparing students for a future where they will likely be programming using AI. P2, P12, and P18 argued that learning computing should not be about programming but should rather be about how to use software tools to solve real problems; up until now writing code has been the most expressive way to do so, but if using AI to generate code becomes the industry norm, then that is what we should be teaching. P17 stated how *"it's inevitable"* that AI tools will pervade future workplaces, so the sooner her students learn to evaluate the output of those tools, the more prepared they will be for industry. This heavy focus on job preparation was also emphasized by P20, who teaches at a public undergraduate-only school designated as a U.S. minority-serving institution (over 70% of students belonging to a minority ethnic group): P20 mentioned that many of his students are looking for focused vocational job training, so it was his responsibility to keep on top of AI coding trends to help his students remain competitive for jobs.
- **Making one's institution more competitive**: A related motivation for embracing AI was several instructors' desires to make their institution stand out among their peers, which could help attract future students. P11 predicted that many universities will be slow to change in response to AI – if his CS department can be among the first to integrate AI coding tools into its curriculum, then it can emerge as a leader in this field. P16 mentioned that the rise of AI tools is a fun opportunity to update her curriculum and that it is generating excitement amongst others in her department. Thus, if more colleagues adopt these tools, then that might elevate her department's reputation as a leader in pedagogy.
- **Covering more advanced material in CS1/CS2**: P2, P11, P12, P16, and P19 discussed how if students can code using the help of AI tools, then that will enable instructors to cover more advanced material in CS1/CS2. Currently, significant amounts of time are spent teaching the rote mechanics of programming. But if AI tools can automate away those mechanics, then CS1/CS2 can be more about software design, which is usually reserved for more advanced courses. P16 said *"I could imagine having a lot more fun talking about concepts rather than syntax"* and how she felt that AI would allow her to cover a lot more ground in CS1. P19 proposed to get rid of what we now think of as CS1/CS2 and instead have students jump straight into software development courses since AI may be able to take care of the more mundane coding details in the future. He made an analogy to the invention of compilers a few decades ago: *"We don't need to look at 1's and 0's anymore, and nobody ever says, 'Wow what a big problem, we don't write machine language anymore!' Compilers are already like AI in that they can outperform the best humans in generating code."*
- **AI may improve equity and access**: In contrast to Section 5.3 where participants mentioned objections related to equity and access, others felt that AI could be beneficial here. For instance, P12 works at a liberal arts college and teaches introductory programming using Arduino to provide a motivating context for students from art and design backgrounds. She felt that the mechanics of coding was discouraging for many of her students who just wanted to create interesting projects. She mentioned that if AI could generate this low-level Arduino code then students can be more motivated to do more creative design and problem-solving in class. P15 similarly mentioned how writing English prompts for AI tools can be far less intimidating than writing code and having to get all the syntax right, so that could make programming more accessible to a wider range of student backgrounds. P19 was excited by how AI code generation can potentially improve diversity in the software industry by encouraging more diverse types of students to learn computing, including those who are currently discouraged by having to manually write code. He felt that future computing

[5]The week after we interviewed P3, OpenAI announced a new GPT-4 LLM, which can now take images as input and analyze those images to generate relevant code [83].

curricula will be more about fostering good communication (i.e., communicating with AI via natural language prompts and back-and-forth dialogue) rather than the obscure details of programming language syntax and semantics.

Integrating AI tools into current courses: Some participants brainstormed ways that AI tools could integrate into their current courses without requiring any major structural changes.

- **Giving personalized help to students**: P12, P15, and P17 were excited that by doing some prompt engineering [112] one could use AI tools to explain step-by-step how code works, teach relevant concepts on-demand within the context of each student's code, and create extra exercises for students who want more practice. P12 reported that after students complete assignments or labs in her class they never go back to reflect on *why* they might have gotten the answers right or wrong. She tries to provide these explanations during office hours and hopes that AI can help do this for students in the future: *"Imagine you hover over a line of code in your IDE and it explains it to you, that could be great for learners."* P17, who teaches at a liberal arts college, described how she purposely does not use a textbook for CS1 so that she can customize the curriculum for their liberal arts context. However, she finds that students sometimes come to her office hours asking for more examples of a specific concept they are learning. So she wants an AI tool that can generate these additional examples and explanations in response to student questions without needing her to be present.
- **Helping instructors with time-consuming tasks**: P14 has already been using AI tools to generate new variants of his programming assignments. He is doing this so that he can create fresh new variants each term to prevent students from copying the answers from friends who took his course in prior terms. In the past he created variants manually, and it was very time-consuming. But this is something that AI is well-suited to do since it can generate a lot of candidate variants and he can pick the most suitable ones to use. Similarly, P17 currently creates small drills for her students to practice basic programming, akin to basic math drills; she wants to use AI to help her generate lots of these drills to give students more practice. P15 and P18 said that their TAs hand-grade hundreds of programming assignments using rubrics to check for code style best-practices and felt that AI could be trained to do this sort of stylistic grading, which is akin to doing a code review. Lastly, P17 wanted to use AI as a helper at her office hours since her CS1 students often ask the same types of basic questions, so she can let the AI take a first pass at answering those while she spends time on the harder questions.

Designing new AI-embedded course materials: In contrast to the ideas above, which can be retrofitted into existing courses, the following ideas aim to redesign courses with AI tools in mind.

- **Focusing more on code reading and critique**: P9, P14, and P15 propose to shift introductory computing courses more toward reading and critiquing code rather than simply writing code. P9 raised this question during her interview:

  "If these tools are the norm moving forward, the emphasis in intro courses might move from code writing to code reading, code comprehension, and testing. How do you validate what comes out of an AI tool? Does it really meet the expectations that you have? [...] Code comprehension and code reading are going to be super important, like can you trust what code you get out of an AI tool?"

  Similarly, P14 describes how he wants his introductory programming course to turn more into an English or literature class where the emphasis turns to reading, critically analyzing, and editing code that may be produced by AI tools. He made the analogy to teaching students to become good editors rather than just writers. Note that this skill of *code review* (i.e., critiquing others' code) is useful even when collaborating with other humans in the workplace, as P15 noted: *"Yeah I think [code review] is generally going to be a skill for a while, even if someone isn't using Copilot, it's a skill you're going to need when working with others in software engineering."*
- **Creating open-ended design assignments**: P2, P3, P11, P12, P15, and P19 were excited about the prospect of making more open-ended design-based assignments as early as CS1/CS2 rather than waiting until more advanced courses. This can be possible because future students will not be as hindered by the mechanics of writing code if AI can help do it. For instance, P2 mentioned giving data science problems where students can find realistic data sets and analyze them in more creative ways, with AI helping them write code using the Python pandas data analysis library [4]. P15 and P19 wanted a portfolio-based approach to assignments like what occurs in art and design schools: Students could design their own projects and use AI to help them code up a prototype. P3 mentioned that open-ended project grading could be done by probing how well each student understood the process of working with AI, including its strengths and weaknesses: *"Tell me what things you tried to do with the AI and failed, and why. If you just press a button and the AI does it all for you, then you won't have a good story to tell about what challenges and setbacks you faced while doing this."*
- **Having students work collaboratively with AI**: Going one step beyond open-ended assignments, P8, P11, P12, and P19 wanted to design assignments where students have to work collaboratively with AI. For instance, P11 and P12 both proposed algorithm design problems where the student would assume the role of a 'client' and specify what they wanted to make in English. Then the AI would generate code, and the student would test and critique it and pass it back to the AI for the next round of iteration. P19 proposed this sort of collaboration even during exams: AI could generate problems for the student to solve and then help the student along when prompted with clarifying questions. P8 proposed a variation of the think-pair-share [51] classroom activity where a pair of students would try to solve a programming task (e.g., reversing a string) and then prompt an AI to solve it. Then each pair would discuss how their human solutions compare to the AI solution.

These sets of ideas reflect the sentiment that it is inevitable that AI tools will become more widespread, so it is futile to resist [6]. Recall that P11 mentioned how computing departments that adapt fastest to this change will emerge as leaders in the coming decade; he recalled a similar moment in the 1990s when many students first gained widespread internet access. Similar to today, instructors were concerned that students could simply look up all the answers online. But eventually, all schools had no choice but to adapt to the internet, and the departments that first embraced students using the internet had an advantage because they adapted early. P11 believed that a similar outcome could happen for AI tools.

6 REFLECTION AND DISCUSSION

The most unique aspect of our interviews was their timeliness. We conducted them in early 2023, during the first full academic term after ChatGPT's release in late 2022, which was likely the first time that many CS1/CS2 instructors started thinking about what to do with their courses in light of the growing prevalence of AI tools. Thus, our study captures a unique moment in time when instructors have started brainstorming but have not solidified their plans yet.

Due to this timing, participants seemed enthusiastic to talk about this topic because it was already on their minds, regardless of whether they had used AI coding tools. Note that we did *not* purposely recruit instructors who had experience with these tools; we tried to get a diverse sample of CS1/CS2 instructors around the world. As Table 1 shows, many had little experience with AI tools.

During the end of their interviews, several mentioned how our conversation helped them to clarify their own thinking about AI coding tools and that they were curious about the ideas that other study participants came up with. For instance, at the end of his interview, P3 asked unprompted, *"One more thing, when you write the paper, can you email it to me since I am very interested in seeing [what other instructors said]?"* We emailed a draft of the accepted paper to all participants to get any additional feedback they had before finalizing the camera-ready.

A limitation of our participants being personally invested in this topic was that we felt there were anchoring effects [41] in their proposed ideas. Despite us designing our interview prompt to encourage open-ended speculative futures brainstorming (Section 4), all participants anchored their responses to their knowledge of present-day tools such as ChatGPT and Copilot, either obtained through personal experience or from what they hear from colleagues. Thus, some of their ideas about longer-term course changes felt like direct reactions to these tools rather than radically-new notions of what computing education ought to be like in the future.

Relating to ideas from other computing education researchers: In the midst of our interviews, we were made aware of several blog posts and a position paper written by computing education researchers in early 2023. To avoid biasing our interviews and data analyses, we waited until after completing our analyses to read these articles in-depth. Here we reflect on some points of commonality between our study and what they wrote: Our participants' ideas about re-orienting programming education toward critiquing code produced by AI resonates with the Bootstrap Project's blog post [19], which posits that meaningful learning happens when one can verify (or refute) someone else's solution (whether that someone is another human or a machine). Some of our findings also resonate with Ko's pair of blog posts [54, 55], such as participants' concerns about students overrelying on AI tools as shortcuts to bypass learning (Section 5.2) and the opportunity to focus more on specifying requirements for software rather than the mechanics of coding (Section 5.4). Lastly, our study findings add empirical detail to several of the high-level themes that Becker et al. proposed in their position paper [14], such as using AI to give personalized tutoring, to help instructors with time-consuming tasks such as creating exercises, and to re-orient courses more toward code reading, along with concerns about academic integrity (Section 5.2) and ethical objections to AI tools. In addition, we present new ideas such as instructors' varied motivations behind *why* they wanted to either resist or embrace AI tools, along with specific proposed ways to make assignments and exams more 'AI-proof' (Section 5.3).

7 OPEN RESEARCH QUESTIONS

Since we are still early in the adoption curve of AI coding tools, as a community we now have a rare window of opportunity to guide their future usage in effective, equitable, and ethical ways. To work toward this goal, we hope that researchers can investigate some of the relevant open questions that our study findings raised:

- Theory-building – Our participants (even those who want to embrace AI tools) worry that students will become overreliant on AI tools without knowing how they work. Thus, we feel that it is important to build theories about how people believe these tools work. For instance, what mental models do novices currently form both about the code that AI generates *and* about how the AI works to produce that code? How do those novice mental models compare to experts' mental models? And what strategies (if any) do novices and experts currently use to try to validate AI tool outputs for themselves? These questions will be critical for designing techniques to guide novices to form viable mental models so that they can learn to use AI tools effectively.
- Scaffolding novice understanding – Related to above, how can we add pedagogical scaffolds to the outputs of AI tools to help novices understand *how* they are coming up with their code suggestions or explanations? Having the AI "show its work" can potentially help novices to better understand both its capabilities and limitations. Recent lines of research from the HCI and XAI (Explainable AI) communities could provide some inspiration. For instance, the *grounded abstraction matching* technique [64] may be one starting point.
- Tailoring AI coding tools for pedagogy – Current AI coding tools are meant to directly help a programmer write code as quickly and efficiently as possible. However, such directness may not be the best for pedagogy since it gives away the (possibly-right, possibly-wrong) answer without making the learner think deeply. How can we tailor these tools to become better at teaching rather than doing, perhaps integrating pedagogical content knowledge [48] about programming?
- Adapting IDEs for AI-aware pedagogy – Today's IDEs are optimized to streamline code writing, but if future AI-aware

curricula move toward emphasizing skills like code comprehension and critique, how should we redesign IDEs to align with these goals? For instance, how can we design IDEs to discourage students from developing harmful habits, such as overreliance on AI-generated code? And how can pedagogical IDEs nudge students to engage in activities such as reading and critically analyzing AI-generated code?

- Equity and access – Participants brought up how AI tools can both be beneficial and detrimental to these goals. So how can we design curricula that use these AI tools in such a way to work toward greater equity and access? On one hand, current AI systems are criticized for their negative impacts on equity [17, 63]. But if these systems become more widespread then perhaps it is necessary to teach everyone to use them or else a new digital divide [31] may open up between those with and without access to modern AI tools. As P9 shared at the end of her interview: *"My concern would be from an equity perspective, the students who don't know about AI tools would be disadvantaged. So how would you make it so that everyone has the opportunity to use them, from an equitable perspective?"*
- Efficacy studies – How can we tell whether AI tools in introductory courses make students more effective? Can we design controlled experiments where cohorts of CS1/CS2 students receive either AI-enriched or AI-free curricula and track their progress throughout the major? How can we design and sustain these longitudinal studies in an ethical way if it turns out that one condition is significantly better or worse for students? And will it even be possible to enforce a control condition if AI tools become so pervasive that most students are regularly accessing them outside of class?
- Evaluating AI-aware assessments – Can we effectively assess student knowledge if future students collaborate with AI tools on their assignments (and perhaps even on exams)? Our participants suggested a variety of alternative assessment methods such as having students record video explanations. How can we evaluate whether these methods are effective?
- Upper-division computing courses – Our study focused on introductory programming courses, but what about uses of AI tools in upper-division courses where the learning objectives differ? Five of our interview participants mentioned how even though they were opposed to AI in introductory courses, they actually wanted to use them in upper-division courses such as software engineering labs.
- `(programming != computing)` – Related to above, our paper focuses on introductory programming education, but *computing* education encompasses a much broader set of topics and learning objectives. How will AI tools like current and future LLMs [24] affect the many facets of computing education that are not just about writing and running code? What about using AI to teach computing concepts to students who do not necessarily want to become programmers [27, 109]?
- Scaling instruction – It seems plausible for AI tools to help scale instructors' human expertise. For example, in large classes it is impractical for any one instructor to provide in-depth feedback on hundreds of code submissions. If AI could be tailored using an instructor's past feedback, these tools could enable consistent, personalized instruction at scale. How might we enable instructors to tailor AI tools to their own context and desires, and what impact might this have on the way instructors design and deliver their classes?
- Beyond autograded programming assignments – Related to above, currently CS1/CS2 assignments often consist of programming prompts that are graded by autograder software. This is the status quo not necessarily because it is ideal but simply due to how well it scales to large classes. It is hard to scale up open-ended free-response questions that need to be hand-graded by humans. However, modern LLMs show promise in natural language reasoning, so perhaps they can help to both design and assess more interesting assignments that go beyond writing code to pass autograder test cases. But that raises concerns about the ethics of having AI do grading of student assignments, so any such interventions may need to be carefully vetted by instructors.
- Rethinking CS1/CS2 in light of AI tools – Lastly, what if we could redesign CS1/CS2 without following the traditions of the past 50+ years [15] of research and practice in our field? If AI coding tools become more pervasive in the future, what timeless pedagogical themes should still remain the same, and what aspects need to be radically reconsidered? How can we prepare our students for the next 50 years? And what will programmers need to know in the year 2073?

8 CONCLUSION

We presented the perspectives of 20 introductory programming instructors across 9 countries on how they plan to adapt their courses in light of the growing prevalence of AI coding tools. Our study captures a rare moment in time during the first full academic term (early 2023) when these AI tools started becoming widely accessible. We found that in the short-term many planned to take immediate measures to discourage cheating. Then opinions diverged about how to work with these AI tools longer-term, with one side wanting to ban them and continue teaching programming fundamentals, and the other side wanting to integrate them into courses to prepare students for future jobs. We hope these findings along with our open research questions can spur conversations about how to work with these tools in effective, equitable, and ethical ways.

ACKNOWLEDGMENTS

Thanks to all of our interview participants for sharing their time and expertise, to the ICER reviewers for their feedback, and to Majeed Kazemitabaar and Sangho Suh for their feedback. Also thanks to a well-timed Hawaii vacation for giving Philip the much-needed headspace to begin thinking about research directions in this emerging area. This material is based upon work supported by the National Science Foundation under Grant No. NSF IIS-1845900.

REFERENCES

[1] 2023. Anthropic: Introducing Claude. https://www.anthropic.com/index/introducing-claude. Accessed: 2023-05-20.

[2] 2023. Codeium: The modern coding superpower. https://codeium.com/. Accessed: 2023-05-20.
[3] 2023. Introducing GitHub Copilot X: Your AI pair programmer is leveling up. https://github.com/features/preview/copilot-x. Accessed: 2023-05-20.
[4] 2023. pandas - Python Data Analysis Library. https://pandas.pydata.org/. Accessed: 2023-03-15.
[5] 2023. ReplitLM: Guides, code and configs for the ReplitLM model family. https://github.com/replit/ReplitLM/. Accessed: 2023-06-05.
[6] 2023. "Resistance is futile" – Borg – Wikipedia. https://en.wikipedia.org/wiki/Borg. Accessed: 2023-06-02.
[7] 2023. Sourcegraph Cody: Read, write, and understand code 10x faster with AI. https://about.sourcegraph.com/cody. Accessed: 2023-05-20.
[8] Ankur Desai and Atul Deo. 2022. Introducing Amazon CodeWhisperer, the ML-powered Coding Companion | AWS Machine Learning Blog. https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/. Accessed: 2023-03-23.
[9] Anushka Patil and Jonah Engel Bromwich. 2020. How It Feels When Software Watches You Take Tests - The New York Times. https://www.nytimes.com/2020/09/29/style/testing-schools-proctorio.html. Accessed: 2023-03-23.
[10] James Auger. 2013. Speculative Design: Crafting the Speculation. *Digital Creativity* 24, 1 (2013), 11–35.
[11] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, and Quoc Le. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732* (2021). arXiv:2108.07732
[12] John A. Bargh and Tanya L. Chartrand. 2014. The Mind in the Middle: A Practical Guide to Priming and Automaticity Research. (2014).
[13] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA 1, Article 78 (apr 2023), 27 pages. https://doi.org/10.1145/3586030
[14] 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* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759
[15] 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
[16] Brett A. Becker, Amber Settle, Andrew Luxton-Reilly, Briana B. Morrison, and Cary Laxer. 2021. Expanding Opportunities: Assessing and Addressing Geographic Diversity at the SIGCSE Technical Symposium. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 281–287.
[17] 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*. 610–623.
[18] 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
[19] Bootstrap Blog. 2023. What Do Tools like ChatGPT Mean for Math and CS Education? https://bootstrapworld.org/blog/misc/thoughts-on-chat-gpt.shtml. Accessed: 2023-03-23.
[20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 1877–1901.
[21] Joy Buolamwini and Timnit Gebru. 2018. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *Conference on Fairness, Accountability and Transparency*. PMLR, 77–91.
[22] Matthew Butterick. 2022. GitHub Copilot Investigation · Joseph Saveri Law Firm & Matthew Butterick. https://githubcopilotinvestigation.com/. Accessed: 2023-03-23.
[23] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. CodeT: Code Generation with Generated Tests. *arXiv preprint arXiv:2207.10397* (2022). arXiv:2207.10397
[24] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, and Greg Brockman. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021). arXiv:2107.03374
[25] Ruijia Cheng, Ruotong Wang, Thomas Zimmermann, and Denae Ford. 2022. "It would work for me too": How Online Communities Shape Software Developers' Trust in AI-Powered Code Generation Tools. arXiv:2212.03491 [cs.HC]
[26] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/
[27] Parmit K. Chilana, Rishabh Singh, and Philip J. Guo. 2016. Understanding Conversational Programmers: A Perspective from the Software Industry. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 1462–1472. https://doi.org/10.1145/2858036.2858323
[28] Colin B. Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. PyMT5: Multi-Mode Translation of Natural Language and Python Code with Transformers. *arXiv preprint arXiv:2010.03150* (2020). arXiv:2010.03150
[29] Mike Conover, Matt Hayes, Ankit Mathur, Xiangrui Meng, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM. https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm. Accessed: 2023-05-20.
[30] Juliet M. Corbin and Anselm L. Strauss. 2008. *Basics of qualitative research: techniques and procedures for developing grounded theory.* SAGE Publications, Inc.
[31] Rowena Cullen. 2001. Addressing the digital divide. *Online information review* 25, 5 (2001), 311–320.
[32] 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
[33] Paul Denny, Sami Sarsa, Arto Hellas, and Juho Leinonen. 2022. Robosourcing Educational Resources – Leveraging Large Language Models for Learnersourcing. arXiv:2211.04715 [cs.HC]
[34] Nolan Dey, Gurpreet Gosal, Zhiming, Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. 2023. Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster. arXiv:2304.03208 [cs.LG]
[35] Thomas Dohmke. 2022. GitHub Copilot Is Generally Available to All Developers. https://github.blog/2022-06-21-github-copilot-is-generally-available-to-all-developers/. Accessed: 2023-03-23.
[36] Fabrice Bellard. 2023. TextSynth. https://textsynth.com/. Accessed: 2023-03-23.
[37] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, and Daxin Jiang. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *arXiv preprint arXiv:2002.08155* (2020). arXiv:2002.08155
[38] 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
[39] 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)*. Association for Computing Machinery, New York, NY, USA, 97–104. https://doi.org/10.1145/3576123.3576134
[40] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. Incoder: A Generative Model for Code Infilling and Synthesis. *arXiv preprint arXiv:2204.05999* (2022). arXiv:2204.05999
[41] Adrian Furnham and Hua Chu Boo. 2011. A Literature Review of the Anchoring Effect. *The journal of socio-economics* 40, 1 (2011), 35–42.
[42] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A Dialogue Model for Academic Research. Blog post. https://bair.berkeley.edu/blog/2023/04/03/koala/
[43] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends® in Programming Languages* 4, 1-2 (2017), 1–119.
[44] Philip J. Guo. 2018. Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–14.
[45] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. 2008. Design as Exploration: Creating Interface Alternatives through Parallel Authoring and Runtime Tuning. In *Proceedings of the 21st Annual ACM Symposium*

on User Interface Software and Technology. 91–100.

[46] J. Hoffman. 2022. *Speculative Futures: Design Approaches to Navigate Change, Foster Resilience, and Co-Create the Cities We Need*. North Atlantic Books. https://books.google.com/books?id=rdd2EAAAQBAJ

[47] Krystal Hu. 2023. ChatGPT Sets Record for Fastest-Growing User Base - Analyst Note | Reuters. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/. Accessed: 2023-03-23.

[48] Aleata Hubbard. 2018. Pedagogical content knowledge in computing education: a review of the research literature. *Computer Science Education* 28, 2 (2018), 117–135. https://doi.org/10.1080/08993408.2018.1509580 arXiv:https://doi.org/10.1080/08993408.2018.1509580

[49] Dhanya Jayagopal, Justin Lubin, and Sarah E. Chasins. 2022. Exploring the Learnability of Program Synthesizers by Novice Programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 64, 15 pages. https://doi.org/10.1145/3526113.3545659

[50] Arianna Johnson. 2023. ChatGPT In Schools: Here's Where It's Banned—And How It Could Potentially Help Students. https://www.forbes.com/sites/ariannajohnson/2023/01/18/chatgpt-in-schools-heres-where-its-banned-and-how-it-could-potentially-help-students/. Accessed: 2023-03-23.

[51] Mahmoud Kaddoura. 2013. Think Pair Share: A Teaching Learning Strategy to Enhance Students' Critical Thinking. *Educational Research Quarterly* 36, 4 (2013), 3–24.

[52] Eirini Kalliamvakou. 2022. Research: quantifying GitHub Copilot's impact on developer productivity and happiness. GitHub blog – https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/. Accessed: 2023-03-15.

[53] 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 (CHI '23)*. Association for Computing Machinery, New York, NY, USA.

[54] Amy J. Ko. 2023. Large Language Models Will Change Programming … a Lot. https://medium.com/bits-and-behavior/large-language-models-will-change-programming-a-lot-5cfe13afa46c. Accessed: 2023-03-23.

[55] Amy J. Ko. 2023. Large Language Models Will Change Programming… a Little. https://medium.com/bits-and-behavior/large-language-models-will-change-programming-a-little-81445778d957. Accessed: 2023-03-23.

[56] Sophia Krause-Levy, Adrian Salguero, Rachel S. Lim, Hayden McTavish, Jelena Trajkovic, Leo Porter, and William G. Griswold. 2023. Instructor Perspectives on Prerequisite Courses in Computing. 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, 277–283. https://doi.org/10.1145/3545945.3569787

[57] Sarah Kreps, R. Miles McCain, and Miles Brundage. 2022. All the News That's Fit to Fabricate: AI-generated Text as a Tool of Media Misinformation. *Journal of experimental political science* 9, 1 (2022), 104–117.

[58] Sam Lau, Ian Drosos, Julia M. Markel, and Philip J. Guo. 2020. The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–11.

[59] 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)*. Association for Computing Machinery, New York, NY, USA, 563–569. https://doi.org/10.1145/3545945.3569770

[60] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[61] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! arXiv:2305.06161 [cs.CL]

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

[63] Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2021. Towards understanding and mitigating social biases in language models. In *International Conference on Machine Learning*. PMLR, 6565–6576.

[64] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D. Gordon. 2023. "What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. 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 598, 31 pages. https://doi.org/10.1145/3544548.3580817

[65] 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

[66] 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

[67] 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

[68] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. 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)*. Association for Computing Machinery, New York, NY, USA, 37–39. https://doi.org/10.1145/3501709.3544280

[69] Thomas Mahatody, Mouldi Sagar, and Christophe Kolski. 2010. State of the Art on the Cognitive Walkthrough Method, Its Variants and Evolutions. *Intl. Journal of Human–Computer Interaction* 26, 8 (2010), 741–785.

[70] Petros Maniatis and Daniel Tarlow. 2023. Large sequence models for software development activities. Google Research Blog – https://ai.googleblog.com/2023/05/large-sequence-models-for-software.html. Accessed: 2023-06-04.

[71] Julia M. Markel and Philip J. Guo. 2021. Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 502–508. https://doi.org/10.1145/3408877.3432533

[72] Max Schaefer, Sarah Nadi, and Frank Tip. 2023. GitHub Next | TestPilot. https://next.github.com/projects/testpilot. Accessed: 2023-03-23.

[73] Andrew M. McNutt, Chenglong Wang, Robert A. Deline, and Steven M. Drucker. 2023. On the Design of AI-Powered Code Assistants for Notebooks. 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 434, 16 pages. https://doi.org/10.1145/3544548.3580940

[74] Yusuf Mehdi. 2023. Reinventing Search with a New AI-powered Microsoft Bing and Edge, Your Copilot for the Web. https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/. Accessed: 2023-03-23.

[75] Lance A. Miller. 1981. Natural Language Programming: Styles, Strategies, and Contrasts. *IBM Systems Journal* 20, 2 (1981), 184–215.

[76] Samim Mirhosseini, Austin Z. Henley, and Chris Parnin. 2023. What Is Your Biggest Pain Point? An Investigation of CS Instructor Obstacles, Workarounds, and Desires. 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, 291–297. https://doi.org/10.1145/3545945.3569816

[77] Piotr Mirowski, Kory W. Mathewson, Jaylen Pittman, and Richard Evans. 2023. Co-Writing Screenplays and Theatre Scripts with Language Models: Evaluation by Industry Professionals. 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 355, 34 pages. https://doi.org/10.1145/3544548.3581225

[78] Vijayaraghavan Murali, Chandra Maddila, Imad Ahmad, Michael Bolin, Daniel Cheng, Negar Ghorbani, Renuka Fernandez, and Nachiappan Nagappan. 2023. CodeCompose: A Large-Scale Industrial Deployment of AI-assisted Code Authoring. arXiv:2305.12050 [cs.SE]

[79] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. CodeGen2: Lessons for Training LLMs on Programming and Natural Languages. arXiv:2305.02309 [cs.LG]

[80] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv:2203.13474 [cs.LG]

[81] OpenAI. 2022. Educator Considerations for ChatGPT. https://platform.openai.com. Accessed: 2023-03-23.

[82] OpenAI. 2022. Introducing ChatGPT. https://openai.com/blog/chatgpt. Accessed: 2023-03-23.

[83] OpenAI. 2023. GPT-4 Technical Report. https://arxiv.org/abs/2303.08774v2.

[84] OpenAI. 2023. OpenAI Playground. https://platform.openai.com. Accessed: 2023-03-23.

[85] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. https://doi.org/10.48550/arXiv.2203.02155 arXiv:arXiv:2203.02155

[86] 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, 754–768.

[87] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv:2302.06590 [cs.SE]

[88] David N. Perkins and Fay Martin. 1986. Fragile Knowledge and Neglected Strategies in Novice Programmers. In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*. 213–229.

[89] Billy Perrigo. 2023. Exclusive: The $2 Per Hour Workers Who Made ChatGPT Safer. https://time.com/6247678/openai-chatgpt-kenya-workers/. Accessed: 2023-03-23.

[90] Chris Perry and Shrestha Basu Mallick. 2023. AI-powered coding, free of charge with Colab: Google Colab will soon introduce AI coding features using Google's most advanced family of code models, Codey. https://blog.google/technology/developers/google-colab-ai-coding-features/. Accessed: 2023-05-20.

[91] Sundar Pichai. 2023. An Important next Step on Our AI Journey. https://blog.google/technology/ai/bard-google-ai-search-updates/. Accessed: 2023-03-23.

[92] 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 Transactions on Computer-Human Interaction (TOCHI)* (2023).

[93] Reed Albergotti. 2023. Startup Replit Launches a ChatGPT-like Bot for Coders | Semafor. https://www.semafor.com/article/02/15/2023/startup-replit-launches-a-chatgpt-like-bot-for-coders. Accessed: 2023-03-23.

[94] Johan Rosenkilde. 2023. How GitHub Copilot is getting better at understanding your code. https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/. Accessed: 2023-05-20.

[95] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development *(IUI '23)*. Association for Computing Machinery, New York, NY, USA.

[96] Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? Proceedings of the 33rd Annual Conference of the Psychology of Programming Interest Group (PPIG 2022). arXiv:2208.06213 [cs.HC]

[97] 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)*. Association for Computing Machinery, New York, NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[98] Otto Seppälä, Petri Ihantola, Essi Isohanni, Juha Sorva, and Arto Vihavainen. 2015. Do We Know How Difficult the Rainfall Problem Is?. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '15)*. Association for Computing Machinery, New York, NY, USA, 87–96. https://doi.org/10.1145/2828959.2828963

[99] Gaelan Steele. 2022. ChatGPT passes the 2022 AP Computer Science A free response section. https://gist.github.com/Gaelan/cf5ae4a1e9d8d64cb0b732cf3a38e04a. Accessed: 2023-03-15.

[100] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. *arXiv preprint arXiv:1906.02243* (2019). arXiv:1906.02243

[101] Jiao Sun, Q. Vera Liao, Michael Muller, Mayank Agarwal, Stephanie Houde, Kartik Talamadupula, and Justin D. Weisz. 2022. Investigating Explainability of Generative AI for Code through Scenario-Based Design. In *27th International Conference on Intelligent User Interfaces* (Helsinki, Finland) *(IUI '22)*. Association for Computing Machinery, New York, NY, USA, 212–228. https://doi.org/10.1145/3490099.3511119

[102] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpaca: A Strong, Replicable Instruction-Following Model. https://crfm.stanford.edu/2023/03/13/alpaca.html.

[103] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[104] Frank Vahid, Lizbeth Areizaga, and Ashley Pang. 2023. ChatGPT and Cheat Detection in CS1 Using a Program Autograding System. https://www.zybooks.com/research-items/chatgpt-and-cheat-detection-in-cs1-using-a-program-autograding-system/. Accessed: 2023-03-15.

[105] Priyan Vaithilingam, Elena L. Glassman, Peter Groenwegen, Sumit Gulwani, Austin Z. Henley, Rohan Malpani, David Pugh, Arjun Radhakrishna, Gustavo Soares, Joey Wang, and Aaron Yim. 2023. Towards More Effective AI-Assisted Programming: A Systematic Design Exploration to Improve Visual Studio IntelliCode's User Experience. In *Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '23)*. Association for Computing Machinery, New York, NY, USA.

[106] 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 *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 332, 7 pages. https://doi.org/10.1145/3491101.3519665

[107] Sander Valstar, Sophia Krause-Levy, Alexandra Macedo, William G. Griswold, and Leo Porter. 2020. Faculty Views on the Goals of an Undergraduate CS Education and the Academia-Industry Gap. 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, 577–583. https://doi.org/10.1145/3328778.3366834

[108] Sander Valstar, Caroline Sih, Sophia Krause-Levy, Leo Porter, and William G. Griswold. 2020. A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 113–123. https://doi.org/10.1145/3372782.3406277

[109] April Y. Wang, Ryan Mitts, Philip J. Guo, and Parmit K. Chilana. 2018. Mismatch of Expectations: How Modern Learning Resources Fail Conversational Programmers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3174085

[110] Dror Weiss. 2022. Announcing Our Next-generation AI Models. https://www.tabnine.com/blog/announcing-tabnine-next-generation/. Accessed: 2023-03-23.

[111] 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

[112] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. arXiv:2302.11382 [cs.SE]

[113] Simon Willison. 2022. Writing Tests with Copilot. https://til.simonwillison.net/til/til/gpt3_writing-test-with-copilot.md. Accessed: 2023-03-23.

[114] Chloe Xiang. 2023. OpenAI Is Now Everything It Promised Not to Be: Corporate, Closed-Source, and For-Profit. https://www.vice.com/en/article/5d3naz/openai-is-now-everything-it-promised-not-to-be-corporate-closed-source-and-for-profit. Accessed: 2023-03-15.

[115] J.D. Zamfirescu-Pereira, Richmond Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3544548.3581388

[116] Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D Goodman, and Nick Haber. 2022. Parsel: A Unified Natural Language Framework for Algorithmic Reasoning. *arXiv preprint arXiv:2212.10561* (2022).