

Responsible and Equitable Use of AI Code Generators in Computer Science Education

A
Dissertation
Presented to
the faculty of the School of Engineering and Applied Science
University of Virginia

in partial fulfillment
of the requirements for the degree

Doctor of Philosophy
in Systems & Information Engineering

by

Nicholas Gardella

May 2026

APPROVAL SHEET

This
Dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Author: Nicholas Gardella



This Dissertation has been read and approved by the examining committee:

Advisor: Sara Riggs

Committee Member: Matthew Bolton

Committee Member: Raymond Pettit

Committee Member: Tariq Iqbal

Committee Member: Seongkook Heo

Accepted for the School of Engineering and Applied Science:



Jennifer L. West, School of Engineering and Applied Science

May 2026



© 2026 Nicholas Gardella



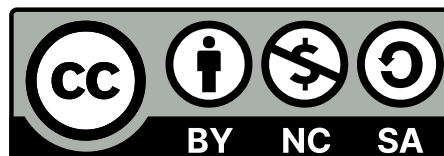
Chapters 2 to 7 specify their own licensing terms which supersede this license. Where contents are not actively covered by another license, the fallback is

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Where no copyright information is provided, CC BY-NC-SA 4.0 applies.

To view a copy of the CC BY-NC-SA 4.0 license, visit:

<https://creativecommons.org/licenses/by-nc-sa/4.0/>



Accessibility Notice

This document is available for free in an audiobook format at <https://doi.org/10.18130/V3/VM1IP0> and on several social media websites across the internet. This PDF was designed with accessibility in mind using PDF standards UA-2 and A-4F, but cannot be updated once published. Please contact the author if you have issues and need a more accessible copy.

for Constance R. English, William T. Scherer, and Zola ❤️



Photo by Todd Wright

Abstract

Generative Artificial Intelligence (AI) is highly effective for software development tasks. GitHub Copilot and other such AI-driven Development Environments (AIDEs) are widely available for students and professional programmers alike. Institutions need to react swiftly to this technology by implementing evidence-based updates to policies and curricula in computer science (CS) education. While research in this area is on the rise, there is limited empirical work that systematically compares individual, AI-assisted, and human-human pair programming paradigms with respect to both well-being and performance over time. Furthermore, despite many attempts to model human interactions with AIDEs, there are several underexplored topics. These include reconciling the numerous and disparate qualitative interaction models, quantifying the relevance of trust to interactions with AIDEs, and exploring implications with respect to educational equity and political economics. This dissertation provides foundational understanding to foster inclusive student success by (1) evaluating and explaining novice-AIDE interactions in terms of performance, well-being, subjective experience, and trust, (2) situating novices' experiences with AIDEs against the societal backdrops of racial exclusion and capitalist economics in education, (3) comparing AI assistance to a social pair-programming alternative, and (4) synthesizing the findings of the prior three steps into a list of evidence-informed recommendations for educational use of code-generating AI.

ACM Computing Classification System Concepts

• Social and professional topics Professional topics Computing education Computing education programs Computer science education • Human-centered computing Human computer interaction (HCI) Empirical studies in HCI • Social and professional topics User characteristics • Social and professional topics ~ Professional topics ~ Computing and business • Applied computing Education • Computing methodologies Artificial intelligence

Keywords

Computer Science Education, Artificial Intelligence-driven Development Environment (AIDE), Introductory Programming, CS1, CS2, AI Code Generators, GitHub Copilot, Novice Programmers, Generative AI, Artificial Intelligence, Trust in Automation, Compliance and Reliance, Software, Human-Automation Interaction, Qualitative, Interview, Diversity, Equity, Capitalism, Critical Political Economy, Dehumanization, Alienation, Theory of Rationalization, Pair Programming, Emotion, Affective Computing, AI-Assisted Programming

Acknowledgments

I am forever grateful for the many who have made this work possible. To my advisor, Sara Riggs, thank you for betting on my success time and again. I threw more than a few curve balls at you, but you kept me consistently on track while offering unparalleled freedom and trust. Your voice in my head will always remind me to articulate the practical importance of my research and, of course, change the word “since” to “given.” Through a hectic five years, you somehow effortlessly got me exactly where you promised. You always stand up for your people, and I am glad to be one of them.

To my committee members, thank you for being the best advocates I could have hoped for. Tariq Iqbal, you helped me challenge my own ideas and offered connectedness in the classroom. Seongkook Heo, your enthusiasm and encouragement helped me take on some of the most challenging of my research accomplishments. Matthew Bolton, your skeptical spirit and paradoxically obsessive yet relaxed demeanor has helped me conquer (some of) my imposter syndrome. Raymond Pettit, you have been a top role model for me; you were an indispensable part of my career development.

To my other collaborators (Joseph Shelton, Isabella Graßl, James Prather, Paul Denny, Juho Leinonen, Ellis Barnes, and Claire Wood), thank you for believing in my work and helping me to grow as a scholar. I owe special thanks to Joseph Shelton as a mentor and to Isabella Graßl and James Prather for building relationships with me out of cold emails. Thank you also to Kylie Gomes for allowing me to participate in her work.

This dissertation would not have been possible without five long years of fellowship support from the University of Virginia and the National Science Foundation (1829004 and 1842490).

Thank you to the many faculty members at the University of Virginia and Virginia State University who enabled my work: Brianna Morrison, Mark Floryan, Rich Nguyen, Xinyao Yi, Brittany Davis, Enoch Solomon, Irene Serrano, and Ahmed Mohammed. Many others are named for their support throughout the acknowledgements of the proceeding chapters.

The $\LaTeX$ compiler will not allow me to list every friend who has been there for me throughout this journey. To my labmates, especially Lauren Horde, it has been a pleasure to have you as friends and colleagues. To my other peers, especially Michael Bertsch and Nora Scheer, I will never forget your kindness and camaraderie. To those who went before, especially Haley Green and Lahiru Nuwan Wijayasingha, you inspired me to push through this work even when I doubted myself. To those not named, you know who you are, and I am deeply grateful for your friendship.

Thank you finally to my family. To my parents, Robert and Lynn, you have exemplified lifelong learning and positive impact in the world. Through every kind of schooling imaginable, you gave everything I needed to succeed (as well as sewing skills and the inability to sit still). To my grandparents, Patricia and Robert, your love, prayers, and hard work laid the groundwork for this legacy. To my siblings, Raymond and Rebecca, thank you for setting strong examples and pushing me down the stairs in a laundry basket. Last but certainly not least, thank you to my spouse, Claire. Our life together is the greatest accomplishment of these past five years. You are my biggest supporter and my best friend.

Contents

Contents	x
List of Figures	xv
List of Tables	xvi
1 Introduction	1
1.1 Problem Area	2
1.2 Scope	4
1.3 Terminology	7
1.4 Related Work	9
1.5 Dissertation Overview	26
References	31
2 Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation	51
2.1 Introduction	52
2.2 Related Work	53
2.3 Method	56
2.4 Results	64
2.5 Discussion	66
2.6 Limitations	68
2.7 Conclusion and Future Work	68

References	69
3 Novice Interactions with GitHub Copilot Under Time Pressure: A Deductive Qualitative Interview Study	74
3.1 Introduction	75
3.2 Method	77
3.3 Related Work	82
3.4 Results	84
3.5 Discussion	123
3.6 Limitations	133
3.7 Future Work	134
3.8 Conclusion	134
References	135
4 Relationships Between Trust, Compliance, and Performance for Novice Programmers Using AI Code Generation	142
4.1 Introduction	144
4.2 Method	150
4.3 Results	159
4.4 Discussion	163
4.5 Conclusion	168
4.6 Key Points	169
References	169
5 HBCU Student Perspectives on Identity, Persistence, and Code-Generating AI in CS Education: A Case Study	177
5.1 Introduction	178
5.2 Positionality Statement	180
5.3 Theoretical Framework	180

5.4	Related Work	182
5.5	Method	185
5.6	Results	189
5.7	Threats to Validity	201
5.8	Discussion	202
5.9	Recommendations	205
5.10	Conclusions and Future Work	207
	References	207

6	Beyond Efficiency: A Critical Political Economy Perspective on AI Code Generators in Computing Education	215
6.1	Introduction	217
6.2	Theoretical Framework	220
6.3	Related Work	227
6.4	Method	231
6.5	Results	236
6.6	Discussion	249
6.7	Educational Implications & Recommendations	259
6.8	Threats to Validity	262
6.9	Conclusion	263
	References	263

7	Fast and Forgettable: A Controlled Study of Novices' Performance, Learning, Work- load, and Emotion in AI-Assisted and Human Pair Programming Paradigms	274
7.1	Introduction	276
7.2	Related Work	277
7.3	Method	282
7.4	Results	290
7.5	Discussion	298

7.6 Threats to Validity	303
7.7 Conclusion	305
References	306
8 Conclusion	316
8.1 Dissertation Summary	318
8.2 Contributions	321
8.3 Educational Recommendations	331
8.4 Limitations	342
8.5 Future Work	350
8.6 Closing Thoughts	352
References	353
Appendix A Interview Guide for Chapter 3	A-1
Appendix B Qualitative Codebook for Chapter 3	B-1
B.1 Interaction Patterns	B-1
B.2 Interaction Quality	B-4
B.3 Work Allocation	B-5
B.4 Appraisals	B-8
B.5 Anthropomorphism	B-9
B.6 Prior Knowledge	B-10
Appendix C Qualitative Reporting Frequencies for Chapter 3	C-1
Appendix D Supplemental Material for Chapter 5	D-1
D.1 Pre-study Questionnaire	D-1
D.2 Exit Interview Guide	D-2
Appendix E Preprints for Chapters 4 and 7	E-1
E.1 Preprint for Chapter 4	E-1

E.2 Preprint for Chapter 7	E-1

List of Figures

1.1	Three participants cohorts studied for this dissertation: Cohorts i, j, and k.	27
2.1	Experimental timeline for each trial that was repeated four times (x4).	60
2.2	Exact questions asked to participants about their emotional affect alongside a complementary visualization.	62
2.3	Decision tree for choosing statistical tests.	63
3.1	Interface seen by participants as they worked on Python programming tasks. . . .	79
3.2	Idea map of qualitative framework resulting from our analysis.	86
4.1	Expected cycle of trust, compliance, and performance from automation trust literature.	146
4.2	Timeline of trust scale administration amidst AI-assisted Python programming. . .	153
4.3	SEM path diagram for the adapted Muir scale.	155
4.4	Boxplots of compliances at each possible score by trial.	161
4.5	Histogram of scores by trial.	164
7.1	Experimental design and counterbalancing permutations.	285
7.2	Experimental procedure.	287
7.3	Performance plots.	292
7.4	Workload and emotion plots.	297

List of Tables

1.1	Mapping of aims to chapters and data sources.	28
1.2	Research questions for Aim 1.	29
1.3	Research questions for Aim 2.	29
1.4	Research questions for Aim 3.	30
2.1	Descriptive statistics for changes in measures from solo trials to trials with the AIDE.	64
2.2	Statistical comparisons between solo and with AI trials for each measure.	65
2.3	Descriptive statistics comparing the participants' first and second trial scores and self-grades for the solo and with AI conditions.	65
2.4	Statistical comparisons between the first and second trial scores and self-grades for the solo and with AI conditions.	66
3.1	Summary of qualitative human-AIDE interaction studies.	85
3.2	Qualitative framework summary.	88
3.3	Summary of prior work and data supporting interaction pattern themes.	91
3.4	Summary of prior work and data supporting interaction quality themes.	102
3.5	Summary of prior work and data supporting work allocation themes.	107
3.6	Summary of prior work and data supporting appraisal themes.	112
3.7	Summary of prior work and data supporting anthropomorphism themes.	117
3.8	Summary of prior work and data supporting prior knowledge themes.	120
4.1	Subjective trust scale adapted from the Muir scale.	154

4.2	Exchangeable Poisson GEE estimates relating pre-trial trust to compliance count.	160
4.3	Exchangeable Poisson GEE estimates relating compliance count to performance score.	162
4.4	Autoregressive gaussian GEE estimates relating performance score to post-trial trust.	163
5.1	Summary of themes in participants' quotes.	190
5.2	Demographics and backgrounds of participants.	191
6.1	Qualitative codebook summary.	237
7.1	Regression coefficients of treatment on performance.	292
7.2	Regression coefficients of treatment on retest performance (learning).	294
7.3	Regression coefficients of treatment on subjective workload.	295
7.4	Regression coefficients of treatment on emotion.	298
8.1	Overview of computing education recommendations developed for Aim 4.	332
C.1	Reporting frequency of direct quotes by participant × topic in Chapter 3	C-2
C.2	Reporting frequency of direct quotes by reference × topic in Chapter 3	C-2

Chapter 1

Introduction

Chapter Contents

1.1	Problem Area	2
1.2	Scope	4
1.2.1	Why Adopt Code Generators?	4
1.2.2	Why Educate Programmers Anymore?	5
1.3	Terminology	7
1.4	Related Work	9
1.4.1	Generative AI for Programming Applications	9
1.4.2	Pair Programming	10
1.4.3	Human-AI Programming	10
1.4.4	GenAI in Computing Education	14
1.4.5	Well-Being	21
1.4.6	Trust	23
1.4.7	Summary of Research Gaps	25
1.5	Dissertation Overview	26
1.5.1	Aim 1- Evaluate & Explain Human-AIDE Interaction	27
1.5.2	Aim 2- Situate Human-AIDE Interaction in a Societal Context	28

1.5.3	Aim 3- Compare Human-AIDE Interaction to a Social Alternative	30
1.5.4	Aim 4- Synthesize Findings Into Computing Education Recommendations .	30
	References	31

1.1 Problem Area

This work investigates human-AI interaction in the programming domain to inform computing pedagogy. Modern transformer-based Large Language Models (LLMs) [153] have enabled the proliferation Artificial Intelligence (AI) code generation tools such as OpenAI Codex [25], Google AlphaCode [86], Amazon CodeWhisperer [31], and Salesforce CodeGen [103]. These technologies stand to change the role of humans in the future of software development [1, 144]. While computing occupations are expected to grow in coming years [23, 40], the Bureau of Labor Statistics predicts that automation will reduce demand for traditional computer programming in favor of professions at higher levels of abstraction from code [23]. This presents a critical need for computer science (CS) education that prepares students to work collaboratively with AI teammates in their careers [81]. One of the most popular and accessible tools employing generative AI technology for programming is GitHub Copilot. Built on the OpenAI Codex model, this Artificial Intelligence-driven Development Environment (AIDE; [35, 123]) was released publicly in June 2022 and has been coined as "your AI pair programmer." Copilot can write multiline, multilingual solutions, match context, write tests, create documentation, and explain code in natural language [46]. At the time of writing, verified students of at least 13 years of age can access Copilot for free through the GitHub Global Campus program [45]. If Copilot can help students build skills, self-efficacy, and interest in computer science, it may be a valuable tool for improving equity in the field [81, 86].

AIDEs are highly capable in solving simple tasks that novices tend to encounter as they are developing computer programming competencies [25, 38]. This raises concerns that students will overuse AIDEs, fail to learn from educational programming assignments, and be unprepared for safety- and privacy-critical programming domains [14, 81, 124]. Leaders in education have

been calling for policies to address these potential risks [43], as less than 10% of over 400 schools surveyed by UNESCO in 2023 reported having official AI policies [146]. A follow-up in 2025 saw that number nearly double to 19%, with another 42% of institutions actively working to develop policies [147].

In addition to risks, policymakers will also need to consider the potential benefits of AIDEs in education. CS education has been quickly growing to meet increasing demands for computing professionals [8]. However, Black, Hispanic, and female workers continue to be underrepresented in computing [40]. Poor representation and retention of these students in colleges and universities is largely to blame [8]. Introductory computer science (CS1) courses are notoriously daunting and challenging for students [96, 114]. Underrepresented students face additional systemic and social barriers to success in these courses [11, 125, 135, 152, 155]. Difficulty and lack of real-world applicability of coursework are factors that drive students to leave CS [11, 15]. By allowing AIDEs in CS1, coursework can become both easier and more engaging for students [81]. Previous pedagogical attempts to make CS more approachable have been successful in improving self-efficacy [127] as well as attracting female [27, 126], Black [27], and non-major [126] students.

The goal of this dissertation is to evaluate AIDEs in the hands of novice and intermediate programmers in light of educational, psychological, social, and economic factors. This will provide invaluable knowledge for educational policymakers and computer science educators about the impacts of this technology. Previous work has studied many aspects of code-generating AI and its implications in computing and computing education. Unfortunately, much of the knowledge base has relied on observational research and speculative reasoning over experimental human studies. This work aims to address this gap by creating evidence-informed, human-centered recommendations through a series of thoughtfully designed laboratory studies. By considering equity implications and well-being factors, the outcomes presented suggest how AI can be used and restricted to best support an inviting computing profession for students in the age of AI.

1.2 Scope

The human studies conducted for this dissertation entailed novice- and intermediate-level programmers writing Python code under time-pressure in a laboratory setting. Two cohorts of participants ($n_i = 17$; $n_j = 10$) programmed alone and with GitHub Copilot, and one cohort ($n_k = 22$) programmed in human-human teams as well. Beyond dedicated attention to the dynamics of the human-AI paradigm, a central emphasis was comparing the AI-assisted paradigm to traditional solo and human-human pair programming paradigms. Data consisted of repeated-measures quantitative data and qualitative data from interviews and questionnaires. Quantitative measures were both objective—programming performance [96, 132] and AI compliance rates [33]—and subjective—self-perceived performance, workload [54] (i.e., mental, physical, temporal, performance, effort, frustration), and emotional affect [131, 129, 111] (i.e., valence and arousal). Performance, self-perceived performance, workload, and emotion were used to compare paradigms, while trust, compliance, and performance were used to study the AI-assisted paradigm specifically. Qualitative data also focused mainly on the AI-assisted paradigm, exploring the human experience of programming with AI, the educational and equity implications of this experience, and the entanglement of this experience with capitalist economic values. The overarching intention of these investigations was to contextualize and evaluate GitHub Copilot relative to past educational programming norms. Since the scope of this work was crafted for a clear relevance to computing pedagogy, it is important to defend the worthiness of this application area. Specifically, the next section defends why to entertain the adoption of code generators and why to preserve computing education, nonetheless.

1.2.1 Why Adopt Code Generators?

Code generation, also known as program synthesis, is an extremely desirable capability for a computer system for two reasons. First, computer generated code can be useful for its ability to solve domain-specific problems. Second, computer generated code can produce better code generators. With these together, code generation technology can *theoretically* enable autore-

gressive value production systems that exponentially reduce the toil of humans over time.

Programming language code has been built into "applications" [17] for decades. In the case of information systems, software helps to keep and review such records as writing, audio, and imagery. Though, software is also at the heart of numerous disparate technologies—planes, pacemakers, utility services, surgical robots, indoor climate control, and farming equipment. In disciplines besides computing itself, code generation has plenty of worthwhile uses.

Code generators themselves are built from code. Just as advancements for the C language were built using previous versions of the C language [75], code generators can improve themselves. Doing so may require human assistance, but the possibility remains. Further, the assistance of the human may become strictly less necessary as the iteration cycles repeat.

It is foolish to ignore this optimistic perspective because, whether noble or greedy, attempts to chase it are inevitable [132, 160]. In a world where the practical limitations of the modern day are gone, an idyllic vision shows computers that satisfy any want or need that humans may demand without imparting any harm or disrupting any meaningful toiling we should still wish to do.

1.2.2 Why Educate Programmers Anymore?

A continued need to educate computer programmers may seem obvious, but it is unfair to leave this assumption unjustified. The preservation of computing education is motivated by (1) temporary limitations of technology, (2) lag in human values behind technological advancements, and (3) durable human values.

All technology faces true limitations specific to the current time; AI code generators are not yet measurably effective enough to uphold software-driven industries without humans. Telephone operators were crucial skilled professionals for much of the 20th century, but once communications technologies advanced, there was objectively no longer a need for their services. Today's best code generators are fighting for scores above 50% on "SWE-Bench," a benchmark that tests the ability to comparably complete software engineering tasks done previously by a humans in a real software project [136]. Even once the best code generators surpass the best human de-

velopers, the infrastructure and energy requirements of AI-driven systems may bottleneck the autonomy of software production for many years [140]. Technology has historically advanced exponentially after key innovations. What capabilities now fill a data center and a solar field might later be contained to a cubic meter. Whatever technical limitations of AI code generators one can propose, innovation has an answer for.

There is also a delay in human values behind the advancement of technology that draws out the timeline for the phaseout of human programmers. In the simple case of electronic mail (email), today's oldest generations may largely favor the physical postal alternative, though society at large endorses the many clear advantages of the electronic option. Even as a disruptive technology can be justified as superior to an existing solution, humans may not feel ready to adopt it yet. Only with time can the social norms around the acceptability of new technology move toward aligning with any objective superiority. Since both technical and social limitations of the current time will fade, to stop here would argue we need only educate programmers *for now*.

Some human values, though, can remain steadfast over time, encouraging continued human involvement in software production despite an economic irrationality in doing so. There will come a time when code generators have surpassed human programmers and humanity has come to accept that truth. Yet, for the desire to keep humanity engaged with what can be termed *critical* and *meaningful* enterprises, humans will continue to be involved with software production. *Critical* enterprises are those like healthcare, agriculture, and water treatment which are, at least seemingly, needed for life. Whether for lack of "trust" in autonomous systems or for deriving meaning in these life-giving pursuits, humans care to be part of critical enterprises, going against the probable ability for autonomous systems to be more effective without human involvement. *Meaningful* enterprises are those which make being alive worthwhile. The dystopian imagery of Pixar's *WALL-E*¹ showed humans who apparently have all needs met by droves of robots and holograms but miss out on the joys of struggling to live and thrive in a natural environment. The production of art—as comedy, food, music, etc.—could potentially be automated by automatically generated software. Setting aside how the art itself would compare to that manually produced,

¹<https://www.pixar.com/wall-e>

the act of creating the art is a meaningful pursuit for the artist. In the same way, building software is, today, an enterprise which is satisfying and fulfilling to many beyond its, only for now, lucrative position in the labor market.

To recap, we should continue to educate computer scientists as we adopt AI code generators. First, humans will be needed to aid a practical transition toward automated software production and fill in many gaps in its abilities. Second, human values will encourage human involvement in software production during a social transition to accepting autonomous software production. Finally, human desires to engage in life-giving and meaningful pursuits will encourage continued involvement indefinitely.

1.3 Terminology

This dissertation studies GitHub Copilot [46], an AI-powered programming toolset that integrates into a programmer’s code editor. Newer versions of Copilot have a chat-based interface, which is now integral to its feature set and offered as a standalone web application².

GenAI Generative Artificial Intelligence (GenAI) as a concept encompasses various approaches to creating outputs that resemble or represent human intelligence in an inauthentic form. This typically happens by compressing insights from a large body of training data created by humans into a predictive model. This model can then be used repeatedly to produce useful related content based on novel inputs.

LLM In common parlance, *GenAI* refers to transformer-based [153] Large-Language Models (LLMs) and tools that rely on them to function. Modern GenAI is not only language-based, but often multi-modal, accepting and producing non-language data such as audio or imagery. However, LLMs are still typically used as interfaces to other types of models, since language is an effective way to communicate with a human user.

²<https://copilot.github.com/>

IDE Integrated Development Environments (IDEs) are code editors with additional programming-adjacent tools such as job shell terminals, testing interfaces, and file explorers. They typically center around the editing of text-based code files. A very popular free IDE is Visual Studio Code (VS Code)³, which has an ecosystem of extensions that go beyond the basic integrated functionality. For example, GitHub Copilot was first implemented as an extension to VS Code. Being developed by the same parent company, Microsoft, VS Code was also heavily modified to support Copilot's features, which are now largely first-party integrations in the editor that complement the extension-powered features. In fact, in 2026, invitations to use Copilot come activated with the IDE, requiring explicit deactivation to be hidden.

AIDE *Artificial Intelligence-driven Development Environment (AIDE)* is a term dating back at least to 2022 [35, 123]. According to Puryear and Sprint, "AIDEs are programming tools, like code completion, that generate suggested code snippets based on a description of what code should do, such as a comment or the start of a function definition" [123]. Copilot, paired with an IDE of choice, such as VS Code or Eclipse™⁴, is a prime example of an AIDE. However, the rising importance of chat-based interfaces in AIDEs mean that generic chatbots like Google Gemini⁵ [142] and ChatGPT⁶ ought to be considered AIDEs as well, given that programmers are heavily using them for software development tasks [42].

Code-generating AI This term is used interchangeably with AIDE, as AIDEs are typically designed to generate code for programmers rather than only providing explanations or debugging help.

AI Code Generator This term is also used interchangeably with AIDEs. AI that does not generate code may still be helpful to software development and computing education, but unrestricted LLMs are typically capable of generating code by default [21]. This means that many

³<https://code.visualstudio.com/>

⁴<https://eclipseide.org/>

⁵<https://gemini.google.com/>

⁶<https://chatgpt.com/>

LLM-based GenAI tools, such as ChatGPT⁶, are AI code generators even though they are not branded only as tools for programmers.

1.4 Related Work

1.4.1 Generative AI for Programming Applications

Researchers imagined communicating with computers through natural language for decades, but only more recently has technology advanced close to this vision [160, 7, 166, 149, 48, 93]. Following the introduction of transformer-based [153] generative AI (GenAI) to the programming domain around 2020 [21], it gained massive research attention; a number of empirical studies investigated its uses in software development but did not include human participants. These studies provided valuable early insights, but could only speculate regarding implications for human users. The seminal work by the creators of the Codex model evaluated their GPT against a human-generated problem dataset called HumanEval⁷, demonstrating a 30% correctness rate with a single attempt and a 70% correctness rate with repeated attempts [25]. Later work replicated similar results using GitHub Copilot, the AIDE built on top of Codex [163]. A slew of studies have evaluated AI code generation technology in terms of code complexity and multilingualism [102], computational complexity [163], code documentation [76], code security [108], vulnerability repair [109], code repairability [28], and bug repair [122]. Several early studies focused specifically on applications in computing education, but did not include human participants [38, 123, 158, 39]. In these studies, AI-based tools produced code that earned grades of 65% to 95% on educational programming tasks (2022; [123]) and scored around the 75th percentile in both CS1 (2022; [38]) and CS2 courses (2023; [39]).

⁷<https://github.com/openai/human-eval>

1.4.2 Pair Programming

As soon as the utility of AI for programming became readily apparent, AI-assisted software development began to be understood as similar to traditional human-human pair programming [46, 64, 28, 91]. Featured notably in the popular Extreme Programming methodology, pair programming is a practice where one programmer (the driver) types out code while another (the navigator) generates ideas and critiques in real time [13]. Beck and colleagues argue that this method is more efficient than having the developers split tasks, work in parallel, and integrate the products at the end [13]. Pair programming has been validated in education settings for its benefits to student performance and persistence [133, 55, 97, 145, 51, 50]. However, logistical challenges and incompatibilities between partners reduce the effectiveness of this model [133, 55, 133]. As such, AI-assisted programming is an attractive means to overcome these challenges [91]. At the time this dissertation was undertaken, very limited work [80, 64, 91] made efforts to compare human-human and human-AI programming paradigms. Even since, only a small number of studies have investigated this comparison either qualitatively [151] or quantitatively [90, 112, 36]. In general, they have found that AI teammates provide superior, low-stakes assistance and boost motivation but lack social presence, smooth communication, and trustworthiness. Human-human-AI teams feature positive aspects from both team paradigms, but AI may tend to dominate the team [90].

1.4.3 Human-AI Programming

Early Qualitative Studies

This section provides a chronology of the qualitative findings of 11 early papers focused on programmer-AI interaction. Some of the common high-level findings include: AIDEs are useful but also imperfect, prompting AIDEs effectively is challenging, evaluating AI-generated code is a task left to users, and over-reliance on AIDEs occurs despite users' moral stigmas against it.

In 2021, IBM researchers performed a qualitative study focused on using neural machine translation to modernize legacy codebases, a messy and difficult process for software developers at the company [157]. They found that objective evaluation of AI-generated code was the best

way for users to build trust in the AI and that it was useful enough for users to tolerate imperfect outputs. Though the participants did not necessarily care to know how the AIDE worked, this sometimes led them to have sub-optimal mental models.

In 2022, Vaithilingam et al. performed an informal analysis of survey responses after participants programmed with and without the GitHub Copilot AIDE [150]. They found that users preferred the AIDE to standard autocomplete because it saved them time and helped them with getting started on programming tasks even when its code was incorrect. Participants used Copilot in lieu of internet search and sometimes over-relied on it. The authors describe three main challenges for users of Copilot: interpreting and assessing its code, debugging its code, and providing it high quality prompts.

Researchers at Google explored strategies and patterns for AIDE interaction through interviews and self-reported programming videos from employees [67]. They found that while participants prompted the AIDE with natural language, some felt that this was unintuitive and that they needed a special prompting style to effectively communicate with the AIDE. Participants usually used bottom-up problem solving and responded to incorrect code by expanding, reducing, or rerunning their prompts.

Jayagopal et al. studied CS2 students' interactions with five program synthesizers including one AIDE to explore design opportunities for making these tools more usable and learnable for novices [66]. They found users prompted the AIDE through "incidental specification" of their intentions through the text in their editors and spent time reading suggestions rather than blindly accepting them (they were not under time pressure). Challenges described included prompting, managing control over numerous features, and recognizing meaningful progress with the synthesizers.

In 2023, researchers at IBM internally recruited 42 participants with interest in software engineering to use a chat-based AIDE [130]. They found that the conversational design exceeded participants' expectations, leading them to enjoy and desire to use the AIDE. Participants requested entire solutions at once or decomposed tasks prior to requesting assistance on subtasks. The authors argue that participants' mental models positioned the AIDE primarily as a tool, but

participants also applied intelligent "mental capacities" to it.

Barke et al. developed a theory of bimodal interaction with AIDEs by conducting programming observations and semi-structured interviews with 20 participants using GitHub Copilot to add functionality to several provided codebases [10]. In the first mode, *acceleration*, programmers maintain control in a flow where the AIDE helps them carry out their plans rapidly. In the second mode, *exploration*, programmers give control to Copilot by using comments to make requests, but then engage carefully with the code it generates.

Bird and colleagues evaluated GitHub Copilot through a case study with five expert programmers and a survey of several thousand users; the experts were willing to give up some control to gain programming efficiency and the respondents reported feeling more productive as they accepted more recommendations from Copilot [16]. The researchers found developers spent more time reviewing Copilot's code than writing code but questioned whether this method would always be most efficient across varied development contexts.

Scoccia performed a qualitative card sorting analysis about use of ChatGPT⁶ for code generation [42]. The data were 100 social media comments on the popular developer forum Hacker News⁸. They found that commenters had both positive and negative experiences, recognized the importance of effective prompt engineering, placed varying degrees of trust in AI outputs, and had differing opinions about whether AI would drastically impact the software development industry.

In 2024, Prather et al. studied novice interactions with and perceptions of the GitHub Copilot AIDE through observations and interviews [119]. They found that novices had positive impressions of Copilot's utility but were also concerned about over-relying on it. Additionally, the authors stressed the importance of two novel interaction patterns: participants' spending considerable time (1) *shepherding* the AI toward their coding goals and (2) *drifting* between failed attempts to utilize AI generated code.

In a global survey of 410 participants with interest in two popular AIDEs, Liang et al. performed mixed-methods analysis of users' reported motivations, strategies, and challenges using AIDEs.

⁸<https://news.ycombinator.com/>

They found that developers preferred fast and cursory interactions with AIDEs to deeper ones and were discouraged from using them by poor quality outputs and difficulty exerting control in their interactions.

In a qualitative follow up to [72], Kazemitabaar et al. used interaction data of 33 young novice programmers to explore the properties of their interactions with a custom AIDE in a self-paced learning experience [73]. They identified four levels of reliance on the AIDE from full reliance to complete lack of reliance and found several signs of over-reliance in their data. They also found that participants mostly prompted the AIDE with natural language and verified code by running it, testing it, or *tinkering* with it.

Finally, Nam et al. conducted a mixed methods study to evaluate an information support approach that used generative AI to accept either prompts or prompt-less triggers [100]. Participants programmed with the AIDE in a series of programming tasks requiring the use of uncommon external libraries. Much of the study's focus was on quantitative and process-based evaluation of how participants engaged with the peculiarities of the prototype. However, the researchers also reported impressions participants gave via surveys they took after the programming tasks. They found that the prototype's integration of coding context helped to speed up users' "information seeking" compared to internet search, but that effectively prompting the AIDE was challenging.

Early Quantitative Studies

Prior to GitHub Copilot's release, Xu and colleagues exposed 31 participants of various abilities to an early-form AIDE; while participants largely reported neutral or positive experiences, they did not show any significant improvements in task correctness or completion times [161]. Similarly, in the first major human study of interactions with Copilot, 24 intermediate and advanced programmers overwhelmingly preferred Copilot to standard autocomplete, but did not demonstrate concrete performance gains [150]. In another study, 21 students with up to eight years of experience programmed with and without Copilot; researchers concluded that the code written with Copilot was comparable in complexity to code written alone, though static code analysis showed a significantly higher Halstead Difficulty (a measure of the difficulty of code for an author

to write and readers to understand). Eye tracking metrics on 15 participants' data also showed significantly shorter and fewer code token fixations when using Copilot [3].

Kazemitabaar et al. ran one of the first studies of novice programmers that focused specifically on performance or learning outcomes for students using AI code generation. In this study, 69 novices between the ages of 10-17 learned to program either with or without a custom Codex-based AIDE. The students who learned with Codex performed significantly better on their practice assignments, and there was no evidence of reduced learning retention on summative exams, on which they worked unassisted. The AI-assisted students who performed best on their entrance exam also showed significant improvements in learning based on the summative exams, indicating that AIDEs can potentially provide learning benefits to more advanced learners [72].

1.4.4 GenAI in Computing Education

Educators have begun using GenAI within the classroom, but research is mixed regarding its benefits and harms to students. Two primary approaches are developing Tools with Guardrails meant only for use in education and studying the Effects of Commodity Tools in education to better understand how to best introduce them pedagogically. The latter is closer to the perspective of this dissertation, but both warrant dedicated attention.

Tools with Guardrails

Overview For years, conversational learning assistants have been developed for education and computing education specifically [167, 104, 105, 24, 78, 68]. Recent leaps in AI reliability and popularity enabled much more capable tooling that has lasted beyond the concept or prototype phase. This section focuses on a few strong cases: CodeHelp⁹ [88], CodeAid[74]/LearnAid¹⁰, and CS50 Duck¹¹ [89]. Note that other tools exist but are not easily accessible for use beyond the creators' home institutions. For example, CodeTailor [60], which uses AI to generate custom

⁹<https://codehelp.app/>

¹⁰<https://learnaid.io/>

¹¹<https://cs50.ai/chat>

Parsons problems for CS students, does not appear to be shared on GitHub or a public website.

CodeHelp CodeHelp is an "Automated Teaching Assistant for Coding and Computer Science".⁹ [88]. The design elevates the basic prompt-response pattern by focusing on structure and reduction of ambiguity. As an alternative for common help requests to human teaching assistants, the prompt asks for three inputs from the student: a code snippet, an error message, and additional detail about the question or issue. When the tool detects that the user has underspecified their request, it will also ask for clarification before answering. Finally, Learning Tools Interoperability (LTI) support and an administrative dashboard view allow instructors to link their courses to CodeHelp and monitor their students' AI interactions. CodeHelp also uses a bring-your-own AI strategy, so users can make use of the app and its educational customizations, while paying for their own computing resources.

CodeAid In relation to their quantitative [72] and qualitative [73] research in this area, Kazemitabaar and colleagues developed two similar products—the programming assistant CodeAid [74] and the startup company LearnAid¹⁰. These bespoke learning platforms, made with students as intended users, add guardrails, customized web-based interfaces, and curricular specificity to commodity LLMs [83]. CodeAid [74] has six core, text-based features to let students ask general or contextual coding questions, learn more about snippets of code, debug snippets of code, and get pseudocode instructions for writing new code. Importantly, CodeAid uses few-shot learning [21] to encourage the base LLM not to provide direct code suggestions for users. All this means is that the developers insert additional text prompting alongside the user's prompt to instruct the model (at inference-time) how it should behave; the weights and biases of the model itself remain unchanged [21]. One consequence of this was that students who used CodeAid also used ChatGPT⁶ when they wanted to overcome these guardrails, aligning with the fears of interviewed instructors [74].

LearnAid LearnAid is an academic spin-off startup company that provides a web-based AI learning platform for educational institutions. At the time of writing, the company only appears to serve the University of Toronto, the creators' home institution¹⁰. In early 2026, publicized features included a learning module, a practice module, and an assignment module, all of which vaguely resemble an AI chat interface or document-based AI service like NotebookLM¹² [82]. Learning mode gives customized text-based instruction with interactive examples that can show, for example, code-tracing timelines in relation to the content. Practice mode can generate questions based on the course content for students to test their knowledge and get customized feedback based on their answers. Assignment mode allows students to upload their code solutions and get help with debugging, understanding, and improvement. These two tools, CodeAid and LearnAid, represent some of the earliest attempts to translate the concept of an AI tutor into a developed learning platform.

CS50 Duck Harvard's collection of CS50 courses serves hundreds of university students and millions of online learners by teaching introductory topics. Their CS50 Duck chatbot is freely available online¹¹ and provides rate-limited chat oriented toward learning to code [89]. As a cost-saving and spam-preventing measure, the duck has a health meter of ten requests that renew by one request every few minutes. It also refuses to generate code and rejects malicious prompting behaviors designed to thwart its guardrails.

Summary The commonality of these tools is that they all attempt to reinvent GenAI tooling to cut out potentially harmful aspects. Namely, the code generation capabilities of the underlying LLMs are severely limited, but the explanation and scaffolding aspects are amplified. By preventing students from using these tools to do their assignments directly or ask excessive and repeated questions, they encourage more moderate and productive interactions. A major limitation is that students who wish to overcome guardrails can still easily seek commodity AI tools like ChatGPT⁶ and Copilot instead. As a result, other researchers have tried to embrace the full capabilities of

¹²<https://notebooklm.google.com/>

GenAI for learning. For example, Denny et al. developed a tool called Promptly¹³ which serves a new type of programming problem called a "prompt problem" [29, 30]. Here, instead of trying to write code manually, students are not only encouraged to prompt AI for code; they are forced to do so as they are prevented from writing or editing code manually. This is meant to teach them how to effectively communicate their goals, a much more relevant skill for a future where AI far exceeds humans' programming abilities. Along the same lines, Porter and Zingaro released an introductory programming textbook that has students use GitHub Copilot and ChatGPT⁶ from the very beginning [116]. This textbook has been used in real course offerings, and students perform in line with institutional norms even when exams contain AI-disallowed questions [148]. Other more gentle introductions have been suggested to introduce students to AI without centering the course around it [37]. These examples stress the importance of continuing to study off-the-shelf AIDEs in educational contexts, as banning them may be at best unnecessary and at worst actively harmful.

Effects of Commodity Tools

Importance Around the first public release of ChatGPT⁶, discussion and speculation in the research community about the pedagogical implications of GenAI became voluble [16, 22, 81, 20, 165, 120]. Educators began adapting by, for example, increasing the weight of exams over programming assignments, teaching about AI ethics, and thinking about how to use AI to offload instructional labor [81]. Some major concerns include lack of consistency among instructors' policies, unclear expectations for students about the extent to which AI usage is acceptable, new effort for instructors to adapt their courses, and keeping up with rapid advancement of the technology; opportunities include more accessible assistance for students and more creative and innovative educational activities [101, 81]. Though the approaches discussed above are an important part of realizing AI's educational potential, study of commodity tools in the wild is another critical component. The reality is that students will mostly use tools they can access for free or via institutional subscriptions rather than bespoke options like CodeHelp⁹.

¹³<https://github.com/thezyrie/promptly>

Metacognitive Support As mentioned in Section 1.4.3, considerable early research has explored human-AI interaction in the programming domain, even with some special attention to novices. Here, the focus is more narrowly on the effects of AI within computing education settings. In a 2022 study of OpenAI's Codex model, which was used to power GitHub Copilot in the early days, 21 university students worked on an assignment to translate natural language to JavaScript™ code [69]. The students reacted to the AI outputs as either correct or incorrect, despite that the AI was fairly inconsistent and non-deterministic. They also felt they needed to invent a special syntax for prompting, in line with the findings of industry research [67] and later work [59]. They also found that though the AI was imperfect, interacting with it was helpful to stimulate exploration and further thought. The authors attribute this to the friction or conflict between the AI's perspective and that of the student, stemming from preconceived notions and prior knowledge [69]. Their results and explanation align with the findings of [10] and are also similar to an idea proposed in [119] that prompting AI provides a "metacognitive scaffold" similar to the practice of "rubber ducking". The practice entails instructing students to verbalize their thought processes and lines of questioning as if to an inanimate rubber duck. This is useful since students often, for example, discover the answer to their own question while asking a human teaching assistant for help with their work. The original version of Harvard's "Duck Debugger" was actually designed around this idea; before the duck had an AI persona, it was simply a useless chatbot that responded to every prompt with "quack" or "quack quack" [89]. As the roots suggest, *metacognition* is cognition about cognition or an awareness, understanding, and control over a mentally demanding problem-solving process [117]. When metacognition is lacking for a student programmer, they might be effectively solving an incorrectly chosen problem or solving a correctly chosen problem with little awareness of how far they are from completing it [118].

Metacognitive Difficulties As much as AI can provide a scaffolding for metacognition, it can disrupt it just the same. Shoufan conducted a study with 56 CS students who took assessments with the help of ChatGPT prior to learning the relevant course material [138]. The students were capable of arriving at correct answers with the AI's help, but were easily misled when they did not

have a good grounding in the subject area. This partially supports the speculative fears that AI could harm students more than experts [81]. On the other hand, a study of 54 students that measured self-regulated learning behaviors (i.e., metacognition) found no relationships at all with AI use [94]. Even across different types of usage (i.e., accelerating code writing, solving one part of a problem, solving multiple parts of a problem, understanding a problem, or understanding code) there were no significant differences in self-regulation. Qualitative research still upholds risks to metacognition, suggesting that individual differences among students potentially account for inconclusive overall effects. Prather et al. used a think-aloud protocol to study the metacognitive behaviors of 21 CS1 students who programmed with the help of GitHub Copilot and ChatGPT; they found stronger students were accelerated by AI and able to ignore it as needed to stay on track, while struggling students experienced a host of metacognitive difficulties and had an "illusion of competence" [118]. Though seemingly in conflict with some findings in [94], this aligns with other findings therein. Margulieux et al. also observed an association with AI use for struggling students. Those in their study who had low self-efficacy and course grades tended to use AI more often and earlier in the course compared to stronger students [94]. Similarly, research about CS students' help-seeking behaviors finds that AI use is bimodal rather than unimodal; some use AI very often while others use it very little, with a gap between [59]. The idea that AI benefits strong students more than weaker students was also suggested by [72] and by Scoccia [42], who identified how prompting AI in English can be challenging for non-native speakers.

Learning There is not great evidence to conclude AI's overall effect on CS students' learning. AI-inclusive pedagogies have shown anecdotally positive receptions from students and instructors [89, 148, 165, 121], yet deliberate studies certainly do not agree conclusively [72, 162, 70]. Kazemitabaar found no significant differences in learning between and AI-assisted and an unassisted learning condition. Though instructive, such an inconclusive finding can often be misinterpreted as a finding of nil effect when it is really a failure to find an effect [72]. Similarly, Xue et al. concluded that ChatGPT does not present a threat to student learning based on lack of significant differences on closed-book post-tests [162]. One study that did show a significant difference was

actually a harm rather than a benefit, but their statistical method was fairly non-conservative, reporting a one-sided Spearman correlation with a p -value of .026 as the marker of statistical significance [70]. Yilmaz and Yilmaz found benefits instead, showing students reporting higher subjective ratings of their computational thinking skills [164]. This would certainly be the case if students were incorrect about their own competence based on metacognitive conflation of their own work to that of their AI assistant [118]. It is worth noting that the authors of [164] advertised in the paper that they were "willing to share their data, analytics methods, and study materials with other researchers upon request," but, after over eight months, have not responded to such a request made via email in June 2025. Ultimately, multiple studies agree that individual differences in how and for what students choose to use AI assistants has a large bearing on learning, with complete outsourcing [119] of coding and critical problem-solving being problematic and explanations and moral support being productive [6, 73, 70, 121].

Self-Efficacy AI also has questionable effects on self-efficacy, a student's belief in their own future ability to succeed, which is generally associated with academic success [58]. As foreshadowed in Metacognitive Difficulties, AI sometimes intrudes [66] and confuses programmers [10, 119], reducing perceived control, something programmers tend to like to maintain [87, 150]. AI also changes how programmers approach the programming process [10, 130, 87], reducing their autonomy over their actions. Like how programmers try to conform to a syntax they find to perform best [67, 69], Hou et al. reported one participant giving the same prompt to the same AI multiple times, considering common aspects of the outputs more reliable [59]. These types of heuristic adaptations provide a false sense of confidence and hinge on the true reliability of a single model. This takes away the programmer's autonomy, as they depend on the AI being correct for their approach to be successful. Research has also found that programmers blame themselves for AI's failures [59, 100, 157], raising concerns about harms to self-efficacy. Though [164] found AI had a positive effect on self-efficacy and motivation, the findings of [94] and [118] suggest that students' self-appraisals may have been misplaced and inappropriately overconfident. Even so, additional confidence could potentially be beneficial to academic success [58].

1.4.5 Well-Being

Stepping back to AI's effects on programmers more broadly, it is very important to consider factors beyond human performance in the form of programming efficiency and academic achievement. As an engineering dissertation, this work draws from formal recommendations by the world's leading engineering professional society, the Institute of Electrical and Electronics Engineers (IEEE). For unfamiliar readers, IEEE has interests across many engineering disciplines, including those outside Electrical Engineering. The IEEE standard 7010-2020 was written to describe recommended approaches for assessing the harms and benefits of autonomous and intelligent systems on human well-being [63]. The standard covers how to assess well-being using "indicators" across 12 "domains." Several of these domains are particularly relevant to research about AI and computing education: the domain of *affect*, the domain of *psychological well-being*, the domain of *community*, and the domain of *work*.

Emotion and Social Connection The domains of *affect* and *community* fall under the broad landscape of emotion [131]. *Affect* refers straightforwardly to positive and negative feelings of various intensities (e.g., calm, happy, bored, angry) [131, 129, 110, 63]. *Community*, on the other hand, is indicated by sense of belonging [63], something that is often explored qualitatively and in conjunction with student identity [143, 125, 65, 32, 57, 128]. Yet, emotions can also have social and community-oriented object foci; one person can feel emotions in relation to another person or group of people [110]. Social connection is foundational to the learning theories of psychologist Lev Vygotsky, who argues that adults and more knowledgeable peers push individuals to advance during childhood development [154]. Ultimately, some combination of the practical and emotional benefits results in the common observation that Pair Programming is mutually beneficial for its participants. In medical education, the stronger teammate may even gain an academic advantage from such "near-peer teaching" [77]. Computing educators recognize the importance of emotional experiences to academic success [49, 47, 44, 18], but only a handful of studies [164, 36, 94] have evaluated code-generating AI using quantitative affective measures.

Self-Efficacy IEEE 7010-2020 distinguishes *psychological well-being* from overall *satisfaction with life* and recommends it be measured by indicators about how worthwhile someone's life pursuits feel to them and how capable they feel at carrying them out [63]. In the present context, psychological well-being is best indicated by measures of self-efficacy (future-oriented; e.g., [139]) and self-perceived performance (past-oriented; e.g. a self-grade). Based on the Scope of the studies undertaken here, self-efficacy is unrealistic to measure repeatedly, as it tends to change slowly (over weeks, months, or years) [94, 164, 58], whereas reflective self-grading can be easily performed on multiple laboratory experiences in a single-day. Self-efficacy also bleeds into the domain of *work*, which can be indicated by environmental measures such as sense of work autonomy [63]. Self-efficacy is also somewhat affective in nature, relating to students' emotions [9, 110] as they undertake challenging coursework [96, 26]. As noted in Self-Efficacy, there is not a consensus about whether code-generating AI is overall good or bad for student self-efficacy.

Workload IEEE 7010-2020 discusses *work* in relation to the mental and physical demands placed on an individual to do activities like homemaking, volunteering, and paid jobs. Beyond autonomy, noted above, the environmental workspace indicators suggested include quantity of working hours and sense of interest in one's work [63]. Though not explicitly mentioned, workload would be another obviously useful indicator at the individual level. *Workload*, as discussed in the human factors literature, is the human cost of work—the toll that performing well on a task takes on the human [54, 53]. Workload is often considered in human factors research because excessive workload can degrade performance, so interventions that reduce workload can support effective interactive technologies. A distinct but related concept, more popular in education psychology, is cognitive load [141]. Workload measures such as the NASA Task Load Index (TLX) [54] can be used to quantify cognitive load [41, 99, 79, 159], but they also bridge beyond the cognitive. In addition to mental demand, effort, and temporal demand, TLX also covers self-perceived performance, physical demand, and frustration, an emotionally charged feeling of discouraging difficulty [18]. Cognitive load is similar to workload in that excessive load,

especially when it is from an *extraneous* or non-essential source, is undesirable. However Cognitive Load Theory (CLT) posits that in a learning context, there must be some load *intrinsic* to the learning task and some more load *germane* to generalizing that task into reusable skills and knowledge [141, 99]. For this reason, too little load can actually be harmful as it can remove a stimulus for learning [12, 154]. Another side effect of excessively easy work is boredom, a low-energy, negative emotion [12, 110, 18]. Though no established standard exists for optimal workload in learning contexts, research on motor learning supports that a moderate level is ideal [2]. Research has found that educational programming tends to stimulate mental demand and effort most compared to other dimensions of the TLX [4]. Other work has suggested that better learning may accompany lower effort and mental demand [115, 99], though again, there could be some point of diminishing returns below which lesser workload could instead be harmful. Very minimal research has explored the quantitative workload effects of AIDEs. One study found that germane cognitive load was significantly higher with a bespoke educational AI tool compared to a human TA; intrinsic and extraneous load were lower but not significantly so [61]. Otherwise, the workload effects of AIDEs are largely underexplored.

1.4.6 Trust

In the human-automation interaction literature, trust—the attitude toward depending on someone or something under conditions of uncertainty—is considered a useful target for optimizing human use of imperfect automation [56, 85]. Here, GitHub Copilot can be likened to other forms of automation, such as industrial control systems [84] and airport security scanners [98]. Copilot offers code suggestions with which programmers can *comply* and now offers suggested code edits on which programmers can *rely*, in that they can choose not to make changes themselves when Copilot is not nudging them to do so [33]. According to theory [85], a programmer’s level of trust in Copilot should regulate how much they choose to depend on its help. Based on their level of dependence and the correctness of Copilot’s suggestions, this feeds into programming performance, which the programmer can observe to adjust their level of trust.

Reliable software is pivotal for a wide variety of safety- and privacy-critical domains including medicine, aerospace technology, and military decision-making. GenAI is not perfectly reliable, so the code it produces can introduce bugs and security vulnerabilities [113, 134, 108] with wide-reaching consequences. Therefore the degree to which programmers rely on and comply with AI suggestions (i.e., dependence or automation use) greatly impacts the effectiveness of a human-AI programming team [56, 85, 107]. If a cycle of trust, compliance, and performance exists in the context of human-AI programming teams, then educational interventions can target programmers' levels of trust to optimize their effectiveness using AI.

A trust-compliance-performance cycle is relatively well-supported by the human factors literature, but little research has studied it in the context of human-AI programming teams. A prior study compared trust between groups of students and found that a younger cohort trusted AIDEs more than an older one [5]. Other researchers found that programmers who wrote more secure code were less trusting than less secure programmers [113]. These findings align with the work discussed in Section 1.4.4 about novices' overconfidence programming with AIDEs. Research about trust's impact on AIDE dependence supports the link qualitatively [66, 113, 130, 150], but quantitative evidence is lacking. For example, participants have been observed accepting code of clearly low-quality [113, 130]. Yet, in many cases, "over-reliance" is labeled as such based on programmers limited engagement with good-quality code [73, 137, 150]. This represents a point of contention between capitalistic, industrial computer science and intellectual or educational computer science. The supposed over-reliance could actually be considered appropriate reliance by the definition of automation misuse [106] as trusting automation in excess of its true capabilities [85]. When one trusts highly in a highly reliable system, it is arguable whether this behavior is inappropriate. In a learning context, there may be a concern, but if one is learning about tasks they may always be able to offload to AI in their career, depending on AI may still be appropriate even when engagement is sacrificed.

1.4.7 Summary of Research Gaps

This section revisits the research gaps identified throughout Sections 1.4.1 to 1.4.6 that this dissertation aims to address. The examples in Section 1.4.4 show that AI has clear pedagogical promise in computing, but it is not enough to study bespoke tools. It is critical to also evaluate and explain how novice programmers use off-the-shelf AIDEs on performance-oriented programming tasks similar to the assessments students face in school. When this work was undertaken, much of the literature looked at the performance of AIDEs without including human participants [25, 102, 163], excluding the all-important influence of human-AI interaction dynamics. The mixed findings of studies that did include humans [72, 161, 150] left it unclear whether or not AIDEs provided a definitive programming performance advantage, especially for novices doing performance-oriented, time-pressured tasks (see Section 1.4.3). Even so, there is insufficient quantitative research about trust in AIDEs to guide educational interventions seeking to balance performance and safety (see Section 1.4.6). The well-being effects of AIDEs are also understood as important to computing education, but quantitative research about them remains quite limited [164, 36, 94, 92, 61]. As for qualitative work, many studies have explored human-AIDE interaction, but the literature is fragmented and biased toward advanced programmers and novelty-sensitive conditions (see Early Qualitative Studies).

Another major gap is the lack of evidence-informed societal critique about the introduction of AIDEs to computing education. CS educators are calling for research into the equity implications of AIDEs [121], but much of the existing work lacks grounding in relevant empirical data [81, 115, 119]. There are also larger sociological discussions to be had about the economic systems that interplay with student perceptions of AI code generators. Though sociological discussions in CS education research have critiqued power structures and explained systems of inequality [34, 62, 19, 71], there is a lack of research using the lens of Critical Political Economy (CPE) [52, 95, 156] to unpack the economic entanglement of software industries with education pipelines.

Finally, the exchangeability of AIDEs with human programming partners is majorly underexplored. The idea that AI-assisted programming is comparable to pair programming is not a new

idea [46, 64, 28, 91], but empirical comparisons came much more recently [90, 112, 36]. At the time this work was undertaken, research that evaluated AI as a substitute for a human partner was very limited [80, 64, 91]. It is known that human-human pair programming is beneficial compared to solo programming for students [133, 55, 97, 145, 51, 50], so it is very important to make an empirical comparison if AI is to be deployed like human assistance in educational settings. Though performance and affective measures have now been addressed by some work [90, 112, 36], none of these studies showed significant effects on learning retention. Measuring of learning differences to a known effective alternative, is an unavoidable need for advising CS education policy.

1.5 Dissertation Overview

The overall goal of this dissertation is to provide foundations for responsible and equitable use of code-generating Artificial Intelligence in computing education. Three arms of questioning are synthesized into a set of educational recommendations for a total of four aims. The remainder of this dissertation consists of seven chapters organized within these four aims:

- **Aim 1:** evaluate what perceptions, behaviors, and outcomes result for novice programmers when they use code-generating AI under time-pressure (Chapters 2 to 4),
- **Aim 2:** situate novice experiences with code-generating AI against equity implications and sociological critiques (Chapters 5 and 6),
- **Aim 3:** interrogate the human-AI programming paradigm against the human-human alternative (Chapter 7)
- **Aim 4:** synthesize a set of evidence-informed recommendations for the use of code-generating AI in computing education (Chapter 8).

The following research question summarizes the overall intent of this work.

Overall Research Question What does studying novice programmers in solo, AI-assisted, and human pair programming paradigms reveal about how computing pedagogy can best promote student success in the age of AI?

The next sections detail the role of each aim and the individual chapters organized within. Figure 1.1 summarizes the 49 participants studied for this work. For clarity, Table 1.1 shows where data from each cohort was used across the four aims. Cohorts i and j were conducted using a very similar methodology with an AI-assisted programming condition and a solo programming control condition, while Cohort k had an AI-assisted condition and a human-human pair programming condition as the control.

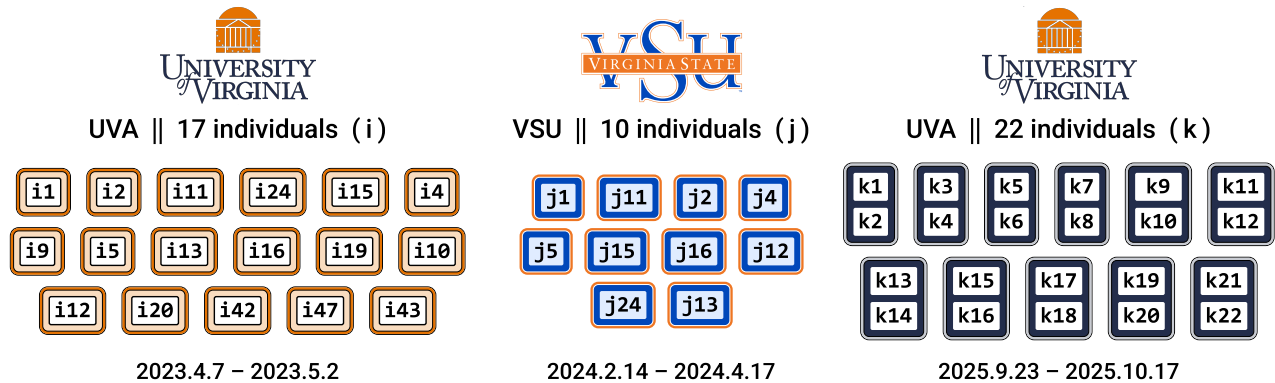


Figure 1.1: Three participants cohorts studied for this dissertation: Cohorts i, j, and k.

1.5.1 Aim 1- Evaluate & Explain Human-AIDE Interaction

The first aim of this work is to evaluate AIDEs for use by novice programmers and describe the qualitative experience and quantitative trust dynamics. Chapter 2 describes a study of Cohort i (see Figure 1.1) evaluating the performance, workload, emotion, and self-efficacy effects of GitHub Copilot compared to a solo programming control condition. Chapter 3 builds on this work by reconciling qualitative insights from the large but fragmented literature discussed in Section 1.4.3 with interviews taken from Cohort i. Chapter 4 draws on quantitative trust, compliance,

Table 1.1: Mapping of aims to chapters and data sources.

Aim	Chapter	Cohorts	Method
1	2	i	Quantitative
	3	i	Qualitative
	4	i + j	Quantitative
2	5	j	Qualitative
	6	i + j	Qualitative
3	7	k	Mixed
4	8	i + j + k	Synthesis

and performance data collected from Cohorts i and j (see Figure 1.1) to evaluate the utility of trust as an target for educational interventions aiming to influence AIDE usage. Table 1.2 details the research questions explored across the three chapters of Aim 1.

1.5.2 Aim 2- Situate Human-AIDE Interaction in a Societal Context

The second aim of this work is to situate novices' experiences programming with GitHub Copilot against the societal backdrops of educational equity and political economy. This approach connects the narrower scope of Aim 1 to a broad social context that considers U.S. computing education in relation to economic value systems and systems of identity-based exclusion. To consider racial exclusion, Chapter 5 details a qualitative case study of the Historically Black College or University (HBCU) students in Cohort j (see Figure 1.1) focused on the nexus of personal identity, persistence in CS education, and the influence of code-generating AI. As for Critical Political Economy, Chapter 6 describes a set of implicit capitalist economic values revealed by interviews with Cohorts i and j, exploring their roots and implications for computing education. Table 1.3 lists the research questions explored in the two investigations under Aim 2.

Table 1.2: Research questions for Aim 1.

<i>Chapter 2</i>	
2.RQ1	How do AIDEs affect the performance, workload, emotion, and self-efficacy of novice programmers under time pressure?
2.RQ2	How do AIDEs influence the effects of additional time spent programming on novices' performance and self-efficacy?
<i>Chapter 3</i>	
3.RQ1	How do novice programmers and AIDEs interact while programming under time pressure?
3.RQ2	What factors can explain how novices use AIDEs while programming under time pressure?
3.RQ3	How does using an AIDE under time pressure affect a novice's thoughts and feelings?
<i>Chapter 4</i>	
4.RQ1	Does specific trust in an AIDE predict subsequent compliance?
4.RQ2	Does compliance predict performance when coding with an AIDE?
4.RQ3	Does programming performance predict subsequent specific trust in an AIDE?

Table 1.3: Research questions for Aim 2.

<i>Chapter 5</i>	
5.RQ1	How do HBCU CS students view their personal and computing identities?
5.RQ2	What factors do HBCU CS students perceive to be helpful and harmful to persistence in computing?
5.RQ3	How do HBCU CS students think about AIDEs in relation to identity and persistence in computing?
<i>Chapter 6</i>	
6.RQ	How can novice programmers' thoughts and feelings about using GitHub Copilot to code be explained by the entanglement of the United States Computer Science education system with capitalist economic values?

1.5.3 Aim 3- Compare Human-AIDE Interaction to a Social Alternative

The third aim of this work builds on the findings from Aims 1 and 2 by comparing the assistance of GitHub Copilot to human-human pair programming, giving special attention to learning and emotion. Due to the educational hazards of AIDEs, it is not enough to compare them to solo programming; rather, they must be compared directly to pair programming, which is a relatively safe and effective alternative. The findings of from Aims 1 and 2 alluded to a lack of social connection between novices and AI programming assistants. Chapter 7 further investigated this issue using a second study design and data from Cohort k (see Figure 1.1). Table 1.4 lists the research questions explored for Aim 3 (Chapter 7).

Table 1.4: Research questions for Aim 3.

Chapter 7	
7.RQ1	To what extent does actual and perceived programming performance differ between the human-human and human-AI team paradigms during a learning effort?
7.RQ2	To what extent does learning retention from a programming effort differ between the human-human and human-AI team paradigms?
7.RQ3	To what extent does the workload (i.e., mental, temporal, effort, frustration) associated with educational programming differ between the human-human and human-AI team paradigms?
7.RQ4	To what extent does the emotional impact of educational programming as measured by emotional valence and arousal differ between the human-human and human-AI team paradigms?

1.5.4 Aim 4- Synthesize Findings Into Computing Education Recommendations

Finally, the Conclusion (Chapter 8) synthesizes all the findings and recommendations from Aims 1, 2, and 3 into a cohesive set of evidence-informed guidelines for using AIDEs like GitHub Copilot in computing education. This discussion refers back to the individual evidence provided in Chapters 2 to 7 and describes how educators can leverage this foundational work to promote

the well-being, academic success, and professional integrity of their students. This Aim ties back the six investigations of the earlier chapters to the overall intent of this dissertation, reiterated below.

Overall Research Question What does studying novice programmers in solo, AI-assisted, and human pair programming paradigms reveal about how computing pedagogy can best promote student success in the age of AI?

Summary The next seven chapters build toward educational recommendations for using AIDEs in CS education (Aim 4). Aim 1 begins by evaluating and explaining human-AIDE interaction. Aim 2 situates human-AIDE interaction within a broader sociological context. Aim 3 compares AIDE assistance to human-human pair programming. Aim 4 concludes with evidence informed implementation guidelines. The next chapter is the first investigation under Aim 1, providing an initial evaluation of GitHub Copilot for use by novice programmers.

References

- [1] Daron Acemoglu and Pascual Restrepo. 2018. Artificial intelligence, automation, and work. In *The economics of artificial intelligence: An agenda*. University of Chicago Press, 197–236. Retrieved Oct. 3, 2023 from <https://www.nber.org/system/files/chapters/c14027/c14027.pdf>.
- [2] Kazunori Akizuki and Yukari Ohashi. 2015. Measurement of functional task difficulty during motor learning: what level of difficulty corresponds to the optimal challenge point? *Human Movement Science*, 43, (Oct. 1, 2015), 107–117. doi:10.1016/j.humov.2015.07.007.
- [3] Naser Al Madi. 2023. How readable is model-generated code? examining readability and visual inspection of GitHub copilot. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. Association for Computing Machinery, New York, NY, USA, (Jan. 5, 2023), 1–5. ISBN: 978-1-4503-9475-8. doi:10.1145/3551349.3560438.

- [4] Naser Al Madi, Siyuan Peng, and Tamsin Rogers. 2022. Assessing workload perception in introductory computer science projects using NASA-TLX. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*. SIGCSE 2022: The 53rd ACM Technical Symposium on Computer Science Education. ACM, Providence RI USA, (Feb. 22, 2022), 668–674. ISBN: 978-1-4503-9070-5. doi:10.1145/3478431.3499406.
- [5] Matin Amoozadeh, David Daniels, Daye Nam, Aayush Kumar, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Trust in generative AI among students: an exploratory study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024: The 55th ACM Technical Symposium on Computer Science Education. ACM, Portland OR USA, (Mar. 7, 2024), 67–73. ISBN: 979-8-4007-0423-9. doi:10.1145/3626252.3630842.
- [6] Matin Amoozadeh, Daye Nam, Daniel Prol, Ali Alfageeh, James Prather, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Student-AI interaction: a case study of CS1 students. (Oct. 11, 2024). arXiv: 2407.00305[cs]. doi:10.48550/arXiv.2407.00305.
- [7] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases—an introduction. *Natural language engineering*, 1, 1, 29–81. Retrieved May 19, 2025 from <https://www.cambridge.org/core/journals/natural-language-engineering/article/natural-language-interfaces-to-databases-an-introduction/21C30448C70DD4988E6DA0D54205FB56>.
- [8] Computing Research Association. 2017. Generation CS: computer science undergraduate enrollments surge since 2006. (2017). <http://cra.org/data/Generation-CS/>.
- [9] Albert Bandura. 1977. Self-efficacy: toward a unifying theory of behavioral change. *Psychological review*, 84, 2, 191. Retrieved Feb. 20, 2026 from <https://psycnet.apa.org/journals/rev/84/2/191/>.
- [10] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: how programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7, 85–111, OOPSLA1. Number: OOPSLA1.
- [11] Lecia J. Barker, Charlie McDowell, and Kimberly Kalahar. 2009. Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin*, 41, 1, (Mar. 4, 2009), 153–157. doi:10.1145/1539024.1508923.
- [12] Ashok R. Basawapatna, Alexander Repenning, Kyu Han Koh, and Hilarie Nickerson. 2013. The zones of proximal flow: guiding students through a space of computational thinking skills and challenges. In *Proceedings of the ninth annual international ACM conference on International computing education research*. ICER '13: International Computing Education Research Conference. ACM, San Diego San California USA, (Aug. 12, 2013), 67–74. ISBN: 978-1-4503-2243-0. doi:10.1145/2493394.2493404.

- [13] Kent Beck, Cynthia Andres, and O'Reilly Online Learning: Academic/Public Library Edition. 2004. *Extreme Programming Explained: Embrace Change. The XP Ser.* Addison Wesley Professional, Boston, (Nov. 16, 2004). ISBN: 978-0-321-27865-4 0-321-27865-8. <http://RE5QY4SB7X.search.serialssolutions.com/?V=1.0&L=RE5QY4SB7X&S=JCs&C=TC0000074886&T=marc>.
- [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*, 500–506.
- [15] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. 2008. Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. *ACM SIGCSE Bulletin*, 40, 1, (Mar. 12, 2008), 402–406. doi:10.1145/1352322.1352274.
- [16] 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. *Commun. ACM*, 66, 6, (May 24, 2023), 56–62. doi:10.1145/3589996.
- [17] Sérena Bochereau, Alexander Terekhov, and Vincent Hayward. 2014. Amplitude and duration interdependence in the perceived intensity of complex tactile signals. In *Haptics: Neuroscience, Devices, Modeling, and Applications*. Vol. 8618. Malika Auvray and Christian Duriez, (Eds.) Series Title: Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, 93–100. ISBN: 978-3-662-44192-3 978-3-662-44193-0. doi:10.1007/978-3-662-44193-0_13.
- [18] Nigel Bosch and Sidney D'Mello. 2017. The affective experience of novice computer programmers. *International Journal of Artificial Intelligence in Education*, 27, 1, (Mar. 1, 2017), 181–206. doi:10.1007/s40593-015-0069-5.
- [19] Pierre Bourdieu. 1986. The forms of capital. In *The sociology of economic life*. (Third edition ed.). Mark S. Granovetter and Richard Swedberg, (Eds.) Routledge, Taylor & Francis Group, New York London. ISBN: 978-0-429-49433-8.
- [20] Aras Bozkurt. 2023. Generative artificial intelligence (AI) powered conversational educational agents: the inevitable paradigm shift. *Asian Journal of Distance Education*, 18, 1, (Mar. 31, 2023). Number: 1. Retrieved June 24, 2025 from <https://www.asianjde.com/ojs/index.php/AsianJDE/article/view/718>.
- [21] Tom Brown et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901. Retrieved July 27, 2023 from https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html?utm_medium=email&utm_source=transaction.

- [22] Christopher Bull and Ahmed Kharrufa. 2023. Generative AI assistants in software development education: a vision for integrating generative AI into educational practice, not instinctively defending against it. *IEEE Software*. Retrieved Oct. 5, 2023 from https://ieeexplore.ieee.org/abstract/document/10213396/?casa_token=DU-frVbLm8MAAAAA:BaVpPE7zkNeaUiAd4Tx_krxU0gIdqQV0RpVYIOhKIWIPK66cwr bxUIc7h5boQhowcMHTHNe4RKo.
- [23] Bureau of Labor Statistics. 2022. Occupational outlook handbook, field of degree: computer and information technology. U. S. Department of Labor. (Sept. 2022). Retrieved July 24, 2023 from <https://www.bls.gov/ooh/field-of-degree/computer-and-information/computer-and-information-technology-field-of-degree.htm>.
- [24] Gustavo Carreira, Leonardo Silva, António José Mendes, and Hugo Gonalo Oliveira. 2022. Pyo, a chatbot assistant for introductory programming students. In *2022 International Symposium on Computers in Education (SIIE)*. 2022 International Symposium on Computers in Education (SIIE). (Nov. 2022), 1–6. doi:10.1109/SIIE56031.2022.9982349.
- [25] Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- [26] Jacqui Chetty and Duan van der Westhuizen. 2013. “i hate programming” and other oscillating emotions experienced by novice students learning computer programming. In *EdMedia*. Association for the Advancement of Computing in Education (AACE), 1889–1894. Retrieved Mar. 5, 2026 from <https://www.learntechlib.org/p/112226/>.
- [27] James P. Cohoon and Luther A. Tychonievich. 2011. Analysis of a CS1 approach for attracting diverse and inexperienced students to computing majors. In *Proceedings of the 42nd ACM technical symposium on Computer science education (SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, (Mar. 9, 2011), 165–170. ISBN: 978-1-4503-0500-6. doi:10.1145/1953163.1953217.
- [28] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub copilot AI pair programmer: asset or liability? *Journal of Systems and Software*, 203, (Sept. 1, 2023), 111734. doi:10.1016/j.jss.2023.111734.
- [29] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt problems: a new programming exercise for the generative AI era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 296–302. ISBN: 979-8-4007-0423-9. doi:10.1145/3626252.3630909.

- [30] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2023. Promptly: using prompt problems to teach learners how to effectively utilize AI code generators. (July 30, 2023). Retrieved Oct. 4, 2023 from <http://arxiv.org/abs/2307.16364> arXiv: 2307.16364[cs].
- [31] Ankur Desai and Atul Deo. 2022. Introducing amazon CodeWhisperer, the ML-powered coding companion. AWS Machine Learning Blog. Section: Amazon Machine Learning. (June 24, 2022). Retrieved July 26, 2023 from <https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/>.
- [32] Amanda B. Diekman, Mia Steinberg, Elizabeth R. Brown, Aimee L. Belanger, and Emily K. Clark. 2017. A goal congruity model of role entry, engagement, and exit: understanding communal goal processes in STEM gender gaps. *Personality and Social Psychology Review*, 21, 2, (May 2017), 142–175. doi:10.1177/1088868316642141.
- [33] Stephen R. Dixon, Christopher D. Wickens, and Jason S. McCarley. 2007. On the independence of compliance and reliance: are automation false alarms worse than misses? *Human Factors*, 49, 4, (Aug. 1, 2007), 564–572. doi:10.1518/001872007X215656.
- [34] Ron Eglash, Audrey Bennett, Laquana Cooke, William Babbitt, and Michael Lachney. 2021. Counter-hegemonic computing: toward computer science education for value generation and emancipation. *ACM Trans. Comput. Educ.*, 21, 4, (Oct. 15, 2021), 29:1–29:30. doi:10.1145/3449024.
- [35] Neil A. Ernst and Gabriele Bavota. 2022. AI-driven development is here: should you worry? *IEEE Software*, 39, 2, (Mar. 2022), 106–110. Conference Name: IEEE Software. doi:10.1109/MS.2021.3133805.
- [36] Guangrui Fan, Dandan Liu, Rui Zhang, and Lihu Pan. 2025. The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, 12, 1, (Mar. 4, 2025), 16. doi:10.1186/s40594-025-00537-3.
- [37] Amanda S. Fernandez and Kimberly A. Cornell. 2024. CS1 with a side of AI: teaching software verification for secure code in the era of generative AI. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 345–351. ISBN: 979-8-4007-0423-9. doi:10.1145/3626252.3630817.
- [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 (ACE '22)*. Association for Computing Machinery, New York, NY, USA, (Feb. 14, 2022), 10–19. ISBN: 978-1-4503-9643-1. doi: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 (ACE '23)*. Association for Computing Machinery, New York, NY, USA, (Jan. 30, 2023), 97–104. ISBN: 978-1-4503-9941-8. doi:10.1145/3576123.3576134.
- [40] Richard Fry, Brian Kennedy, and Cary Funk. 2021. STEM jobs see uneven progress in increasing gender, racial and ethnic diversity. *Pew Research Center*, 1–28. Retrieved Sept. 25, 2023 from https://www.pewresearch.org/science/wp-content/uploads/sites/16/2021/03/PS_2021.04.01_diversity-in-STEM_REPORT.pdf.
- [41] Peter Gerjets, Katharina Scheiter, and Richard Catrambone. 2006. Can learning from molar and modular worked examples be enhanced by providing instructional explanations and prompting self-explanations? *Learning and Instruction*, 16, 2, 104–121. Retrieved Dec. 28, 2023 from <https://www.sciencedirect.com/science/article/pii/S0959475206000144>.
- [42] Gian Luca Scoccia. 2023. Exploring early adopters' perceptions of ChatGPT as a code generation tool, (Sept. 11, 2023). MAG ID: 4388214696. doi:10.1109/asew60602.2023.00016.
- [43] Stefania Giannini. 2023. Reflections on generative AI and the future of education. (2023). Retrieved July 27, 2023 from <https://unesdoc.unesco.org/ark:/48223/pf0000385877>.
- [44] Daniela Girardi, Nicole Novielli, Davide Fucci, and Filippo Lanubile. 2020. Recognizing developers' emotions while programming. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, New York, NY, USA, (Oct. 1, 2020), 666–677. ISBN: 978-1-4503-7121-6. doi:10.1145/3377811.3380374.
- [45] GitHub. 2023. Apply to GitHub global campus as a student. GitHub Docs. Retrieved July 31, 2023 from <https://ghdocs-prod.azurewebsites.net/en/education/explore-the-benefits-of-teaching-and-learning-with-github-education/github-global-campus-for-students/apply-to-github-global-campus-as-a-student>.
- [46] GitHub. 2023. GitHub copilot · your AI pair programmer. GitHub. Retrieved July 24, 2023 from <https://github.com/features/copilot>.
- [47] Jamie Gorson, Kathryn Cunningham, Marcelo Worsley, and Eleanor O'Rourke. 2022. Using electrodermal activity measurements to understand student emotions while programming. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1. ICER 2022: ACM Conference on International Computing Education Research*. ACM, Lugano and Virtual Event Switzerland, (Aug. 3, 2022), 105–119. ISBN: 978-1-4503-9194-8. doi:10.1145/3501385.3543981.

- [48] Tihomir Gvero and Viktor Kuncak. 2015. Synthesizing java expressions from free-form queries. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. SPLASH '15: Conference on Systems, Programming, Languages, and Applications: Software for Humanity. ACM, Pittsburgh PA USA, (Oct. 23, 2015), 416–432. ISBN: 978-1-4503-3689-5. doi:10.1145/2814270.2814295.
- [49] Patricia Haden, Dale Parsons, Krissi Wood, and Joy Gasson. 2017. Student affect in CS1: insights from an easy data collection tool. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17)*. Association for Computing Machinery, New York, NY, USA, (Nov. 16, 2017), 40–49. ISBN: 978-1-4503-5301-4. doi:10.1145/3141880.3141881.
- [50] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. Pair programming in education: a literature review. *Computer Science Education*, 21, 2, 135–173.
- [51] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I. K. Sjøberg. 2009. The effectiveness of pair programming: a meta-analysis. *Information and Software Technology*. Special Section: Software Engineering for Secure Systems 51, 7, (July 1, 2009), 1110–1122. doi:10.1016/j.infsof.2009.02.001.
- [52] Jonathan Hardy. 2014. *Critical Political Economy of the Media: An Introduction*. (1st ed.). Routledge, (June 20, 2014). ISBN: 978-0-203-13622-5. doi:10.4324/9780203136225.
- [53] Sandra G. Hart. 2006. Nasa-task load index (NASA-TLX); 20 years later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 50, 9, (Oct. 2006), 904–908. doi:10.1177/154193120605000909.
- [54] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (task load index): results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.
- [55] Anja Hawlitschek, Sarah Berndt, and Sandra Schulz. 2023. Empirical research on pair programming in higher education: a literature review. *Computer Science Education*, 33, 3, (July 3, 2023), 400–428. doi:10.1080/08993408.2022.2039504.
- [56] Kevin Anthony Hoff and Masooda Bashir. 2015. Trust in automation: integrating empirical evidence on factors that influence trust. *Human Factors*, 57, 3, (May 1, 2015), 407–434. doi:10.1177/0018720814547570.
- [57] John L. Holland. 1985. *Making Vocational Choices: A Theory of Vocational Personalities and Work Environments*. Prentice-Hall, Englewood Cliff, NJ. ISBN: 978-0-13-547597-3.
- [58] Toni Honicke and Jaclyn Broadbent. 2016. The influence of academic self-efficacy on academic performance: a systematic review. *Educational research review*, 17, 63–84. Retrieved Dec. 29, 2023 from <https://www.sciencedirect.com/science/article/pii/S1747938X15000639>.

- [59] Irene Hou, Sophia Mettill, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The effects of generative AI on computing students' help-seeking preferences. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*. Association for Computing Machinery, New York, NY, USA, (Jan. 29, 2024), 39–48. ISBN: 979-8-4007-1619-5. doi:10.1145/3636243.3636248.
- [60] Xinying Hou, Zihan Wu, Xu Wang, and Barbara J. Ericson. 2024. CodeTailor: LLM-powered personalized parsons puzzles for engaging support while learning programming. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale. L@S '24: Eleventh ACM Conference on Learning @ Scale*. ACM, Atlanta GA USA, (July 9, 2024), 51–62. ISBN: 979-8-4007-0633-2. doi:10.1145/3657604.3662032.
- [61] Anna Y. Q. Huang, Cheng-Yan Lin, Sheng-Yi Su, and Stephen J. H. Yang. 2025. The impact of GenAI-enabled coding hints on students' programming performance and cognitive load in an SRL-based python course. *British Journal of Educational Technology*, 56, 5, 1942–1972. _eprint: <https://bera-journals.onlinelibrary.wiley.com/doi/pdf/10.1111/bjet.13589>. doi:10.1111/bjet.13589.
- [62] Yujeong Hwang, Anjali Das, Jane Waite, and Sue Sentence. 2023. Using a sociological lens to investigate computing teachers' culturally responsive classroom practices. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23)*. Vol. 1. Association for Computing Machinery, New York, NY, USA, (Sept. 10, 2023), 206–221. ISBN: 978-1-4503-9976-0. doi:10.1145/3568813.3600112.
- [63] IEEE. 2020. IEEE recommended practice for assessing the impact of autonomous and intelligent systems on human well-being. *IEEE Std 7010-2020*, (May 2020), 1–96. Conference Name: IEEE Std 7010-2020. doi:10.1109/IEEESTD.2020.9084219.
- [64] Saki Imai. 2022. Is github copilot a substitute for human pair-programming? an empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 319–321.
- [65] Kathleen Isenegger, Kari L. George, Paul Bruno, and Colleen M. Lewis. 2023. Goal-congruity theory predicts students' sense of belonging in computing across racial/ethnic groups. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, (Mar. 3, 2023), 1069–1075. ISBN: 978-1-4503-9431-4. doi:10.1145/3545945.3569834.
- [66] 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*, 1–15.

- [67] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J. Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–19.
- [68] W. Lewis Johnson. 2001. Pedagogical agent research at CARTE. *AI Magazine*, 22, 4, 85–85. Retrieved Mar. 3, 2026 from <https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1594>.
- [69] Martin Jonsson and Jakob Tholander. 2022. Cracking the code: co-coding with AI in creative programming education. In *Proceedings of the 14th Conference on Creativity and Cognition (C&C '22)*. Association for Computing Machinery, New York, NY, USA, (June 20, 2022), 5–14. ISBN: 978-1-4503-9327-0. doi:10.1145/3527927.3532801.
- [70] Gregor Jošt, Viktor Taneski, and Sašo Karakatič. 2024. The impact of large language models on programming education and student learning outcomes. *Applied Sciences*, 14, 10, (Jan. 2024), 4115. Number: 10. doi:10.3390/app14104115.
- [71] Maria Kallia and Quintin Cutts. 2021. Re-examining inequalities in computer science participation from a bourdieusian sociological perspective. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021)*. Association for Computing Machinery, New York, NY, USA, (Aug. 17, 2021), 379–392. ISBN: 978-1-4503-8326-4. doi:10.1145/3446871.3469763.
- [72] 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, (Apr. 19, 2023), 1–23. ISBN: 978-1-4503-9421-5. doi:10.1145/3544548.3580919.
- [73] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–12. ISBN: 979-8-4007-1653-9. doi:10.1145/3631802.3631806.
- [74] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, (May 11, 2024), 1–20. ISBN: 979-8-4007-0330-0. doi:10.1145/3613904.3642773.

- [75] Brian W. Kernighan and Dennis M. Ritchie. 1988. *The C programming language*. Pearson Educación. Retrieved Mar. 2, 2026 from https://books.google.com/books?hl=en&lr=&id=OpJ_0zpF7jIC&oi=fnd&pg=PP11&dq=info:as-mZnbSCAAJ:scholar.google.com&ots=2eN_cg10_&sig=yHYFBR_K8ThLn-9tBTRqz0uKh1s.
- [76] Junaed Younus Khan and Gias Uddin. 2022. Automatic code documentation generation using gpt-3. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 1–6.
- [77] András Komócsi, Gergely Csaba, Eszter Veronika Csöngéi, Krisztina Fischer, Kristóf Filipánits, László Czopf, and Andrea Tamás. 2026. Academic performance and progression among near-peer tutors: a comparative analysis in undergraduate medical education. *Medical Teacher*, 48, 3, (Mar. 4, 2026), 394–404. [_eprint: https://doi.org/10.1080/0142159X.2025.2553658](https://doi.org/10.1080/0142159X.2025.2553658). doi:10.1080/0142159X.2025.2553658.
- [78] Mario Konecki, Nikola Kadoić, and Rok Piltaver. 2015. Intelligent assistant for helping students to learn programming. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). (May 2015), 924–928. doi:10.1109/MIPRO.2015.7160406.
- [79] Thomas Kosch, Jakob Karolus, Johannes Zagermann, Harald Reiterer, Albrecht Schmidt, and Paweł W. Woźniak. 2023. A survey on measuring cognitive workload in human-computer interaction. *ACM Computing Surveys*, 55, 13, (Dec. 31, 2023), 1–39. doi:10.1145/3582272.
- [80] Sandeep Kaur Kuttal, Bali Ong, Kate Kwasny, and Peter Robe. 2021. Trade-offs for substituting a human with an agent in a pair programming context: the good, the bad, and the ugly. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, (May 7, 2021), 1–20. ISBN: 978-1-4503-8096-6. doi:10.1145/3411764.3445659.
- [81] Sam Lau and Philip 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 2023: ACM Conference on International Computing Education Research*. ACM, Chicago IL USA, (Aug. 7, 2023), 106–121. ISBN: 978-1-4503-9976-0. doi:10.1145/3568813.3600138.
- [82] LearnAid AI. 2026. LearnAid AI YouTube channel. (Feb. 25, 2026). Retrieved Mar. 3, 2026 from <https://www.youtube.com/@LearnAidAI>.
- [83] 2025. LearnAid and AI in the classroom. In collab. with Tovi Grossman. Published: CBC/Radio-Canada Radio One 99.1 FM (Toronto). Toronto Ontario Canada, (Oct. 28, 2025). Retrieved Mar. 3, 2026 from

https://www.linkedin.com/posts/tovi-grossman-40430166_this-morning-i-joined-cbcradio-canada-radio-activity-7389063667964751872-0IpA/.

- [84] John Lee and Neville Moray. 1992. Trust, control strategies and allocation of function in human-machine systems. *Ergonomics*, 35, 10, (Oct. 1992), 1243–1270. doi:10.1080/00140139208967392.
- [85] John D. Lee and Katrina A. See. 2004. Trust in automation: designing for appropriate reliance. *Human factors*, 46, 1, 50–80. Publisher: SAGE Publications Sage UK: London, England.
- [86] Yujia Li et al. 2022. Competition-level code generation with AlphaCode. *Science*, 378, 6624, (Dec. 9, 2022), 1092–1097. doi:10.1126/science.abq1158.
- [87] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A large-scale survey on the usability of AI programming assistants: successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–13. ISBN: 979-8-4007-0217-4. doi:10.1145/3597503.3608128.
- [88] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. Koli Calling '23: 23rd Koli Calling International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 13, 2023), 1–11. ISBN: 979-8-4007-1653-9. doi:10.1145/3631802.3631830.
- [89] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024. Teaching CS50 with AI: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 750–756. ISBN: 979-8-4007-0423-9. doi:10.1145/3626252.3630938.
- [90] Wenhan Lyu, Yimeng Wang, Yifan Sun, and Yixuan Zhang. 2025. Will your next pair programming partner be human? an empirical evaluation of generative AI as a collaborative teammate in a semester-long classroom setting. In *Proceedings of the Twelfth ACM Conference on Learning @ Scale*. L@S '25: Twelfth ACM Conference on Learning @ Scale. ACM, Palermo Italy, (July 17, 2025), 83–94. ISBN: 979-8-4007-1291-3. doi:10.1145/3698205.3729544.
- [91] Qianou Ma, Tongshuang Wu, and Kenneth Koedinger. 2023. Is AI the better programming partner? human-human pair programming vs. human-AI pAIr programming. (June 9, 2023). arXiv: 2306.05153 [cs]. doi:10.48550/arXiv.2306.05153.

- [92] Lauri Malmi, Judy Sheard, Päivi Kinnunen, Simon, and Jane Sinclair. 2020. Theories and models of emotions, attitudes, and self-efficacy in the context of programming education. In *Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER '20)*. Association for Computing Machinery, New York, NY, USA, (Aug. 7, 2020), 36–47. ISBN: 978-1-4503-7092-9. doi:10.1145/3372782.3406279.
- [93] Mehdi Manshadi, Daniel Gildea, and James Allen. 2013. Integrating programming by example and natural language programming. In *Proceedings of the AAAI conference on artificial intelligence* number 1. Vol. 27, 661–667. Retrieved May 19, 2025 from <https://ojs.aaai.org/index.php/AAAI/article/view/8695>.
- [94] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-regulation, self-efficacy, and fear of failure interactions with how novices use LLMs to solve programming problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2024: Innovation and Technology in Computer Science Education*. ACM, Milan Italy, (July 3, 2024), 276–282. ISBN: 979-8-4007-0600-4. doi:10.1145/3649217.3653621.
- [95] Karl Marx. 2023. Economic and philosophic manuscripts of 1844. In *Social theory re-wired*. Routledge, 142–147. Retrieved Mar. 11, 2025 from <https://www.taylorfrancis.com/chapters/edit/10.4324/9781003320609-19/economic-philosophic-manuscripts-1844-karl-marx>.
- [96] Michael McCracken et al. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '01)*. Association for Computing Machinery, New York, NY, USA, (Dec. 1, 2001), 125–180. ISBN: 978-1-4503-7359-3. doi:10.1145/572133.572137.
- [97] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. 2002. The effects of pair-programming on performance in an introductory programming course. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 38–42.
- [98] Stephanie M. Merritt and Daniel R. Ilgen. 2008. Not all trust is created equal: dispositional and history-based trust in human-automation interactions. *Human factors*, 50, 2, 194–210. Number: 2 Publisher: SAGE Publications Sage CA: Los Angeles, CA.
- [99] Briana Baker Morrison. 2016. *Replicating experiments from educational psychology to develop insights into computing education: cognitive load as a significant problem in learning programming*. Ph.D. Dissertation. Georgia Institute of Technology. Retrieved Mar. 5, 2026 from https://search.proquest.com/openview/567e395bdceec6d51b03d667887a232d/1?pq-origsite=gscholar&cbl=18750&diss=y&casa_token=Bj5ZfFQVsv4AAAAA:BO-tE8uf8AYS6sdABrmYXHmgQGT96UDdPdEVyY0vyAuaBo0EHK8kNPV7Q3G-Y24Svi39QjI4Do.

- [100] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24: IEEE/ACM 46th International Conference on Software Engineering. ACM, Lisbon Portugal, (Apr. 12, 2024), 1–13. ISBN: 979-8-4007-0217-4. doi:10.1145/3597503.3639187.
- [101] Michael Neumann, Maria Rauschenberger, and Eva-Maria Schön. 2023. “we need to talk about ChatGPT”: the future of AI and higher education. In *2023 IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG)*. 2023 IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG). (May 2023), 29–32. doi:10.1109/SEENG59157.2023.00010.
- [102] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22)*. Association for Computing Machinery, New York, NY, USA, (Oct. 17, 2022), 1–5. ISBN: 978-1-4503-9303-4. doi:10.1145/3524842.3528470.
- [103] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: an open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.
- [104] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. 2020. Python-bot: a chatbot for teaching python programming. *Engineering Letters*, 29, 1. Retrieved Mar. 3, 2026 from <https://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=1816093X&AN=148993113&h=gp3VyxguUsDLWCJHKuQyBz%2BvjnXfyf3GKbJwC52B9EYHySSBj2skGPnTYjYImg0f0lK019I0FlwbppT9cu5VQ%3D%3D&crl=c>.
- [105] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. 2022. Revision-bot: a chatbot for studying past questions in introductory programming. *IAENG International Journal of Computer Science*, 49, 3. Retrieved Mar. 3, 2026 from <https://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=1819656X&AN=158903973&h=Q%2Fr2yg3WVyDRtFk4HZ9gYeTKd8ti0n%2FsHQUqWRwfgLg1xEYvWqrCF%2B5vpno5XyKwJce01QgTAxDEleC39vkxqQ%3D%3D&crl=c>.
- [106] Raja Parasuraman and Victor Riley. 1997. Humans and automation: use, misuse, disuse, abuse. *Human Factors*, 39, 2, (June 1, 1997), 230–253. doi:10.1518/001872097778543886.
- [107] Colleen E. Patton and Christopher D. Wickens. 2024. The relationship of trust and dependence. *Ergonomics*, 67, 11, (Nov. 2024), 1535–1552. doi:10.1080/00140139.2024.2342436.

- [108] 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.
- [109] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2023. Examining zero-shot vulnerability repair with large language models. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2023 IEEE Symposium on Security and Privacy (SP). (May 2023), 2339–2356. doi:10.1109/SP46215.2023.10179324.
- [110] Reinhard Pekrun. 2024. Control-value theory: from achievement emotion to a general theory of human emotions. *Educational Psychology Review*, 36, 3, (Sept. 2024), 83. doi:10.1007/s10648-024-09909-7.
- [111] Reinhard Pekrun. 2006. The control-value theory of achievement emotions: assumptions, corollaries, and implications for educational research and practice. *Educational Psychology Review*, 18, 4, (Nov. 30, 2006), 315–341. doi:10.1007/s10648-006-9029-9.
- [112] Jacob Penney, Pawan Acharya, Peter Hilbert, Priyanka Parekh, Anita Sarma, Igor Steinmacher, and Marco Gerosa. 2025. Understanding programming students' help-seeking preferences in the era of generative AI. In *Proceedings of the ACM Global Computing Education Conference 2025 - Volume 1*. CompEd 2025: ACM Global Computing Education Conference 2025. ACM, Gaborone Botswana, (Oct. 21, 2025), 15–21. ISBN: 979-8-4007-1929-5. doi:10.1145/3736181.3747165.
- [113] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, (Nov. 21, 2023), 2785–2799. ISBN: 979-8-4007-0050-7. doi:10.1145/3576915.3623157.
- [114] Andrew Petersen, Michelle Craig, Jennifer Campbell, and Anya Tafliovich. 2016. Revisiting why students drop CS1. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli Calling 2016: 16th Koli Calling International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 24, 2016), 71–80. ISBN: 978-1-4503-4770-9. doi:10.1145/2999541.2999552.
- [115] Carrie Anne Philbin. 2023. Exploring the potential of artificial intelligence program generators in computer programming education for students. *ACM Inroads*, 14, 3, (Sept. 2023), 30–38. doi:10.1145/3610406.
- [116] Leo Porter and Daniel Zingaro. 2023. *Learn AI-Assisted Python Programming With GitHub Copilot and ChatGPT*. Manning, (Nov. 21, 2023). ISBN: 978-1-63343-778-4.
<https://www.manning.com/books/learn-ai-assisted-python-programming>.

- [117] James Prather. 2018. *Beyond Automated Assessment: Building Metacognitive Awareness in Novice Programmers in CS1*. Ph.D. Dissertation. Nova Southeastern University. Retrieved Mar. 5, 2026 from https://search.proquest.com/openview/9c25fd75a7fe1291e9208c771c46b2fc/1?pq-origsite=gscholar&cbl=18750&casa_token=e9uD9Im8PUIAAAAA:HZSLHrgLnwdriKZFwnqJ1CZ18wi9_I9bSdliRodfK70s-wNbEd-89dfR0iFg2CsPpFOctnVPX60.
- [118] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: the benefits and harms of generative AI for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*. ICER 2024: ACM Conference on International Computing Education Research. ACM, Melbourne VIC Australia, (Aug. 12, 2024), 469–486. ISBN: 979-8-4007-0475-8. doi:10.1145/3632620.3671116.
- [119] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. “it’s weird that it knows what i want”: usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 29, 2024), 1–31. doi:10.1145/3617367.
- [120] James Prather et al. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, (Dec. 2023), 108–159. ISBN: 979-8-4007-0405-5. Retrieved Sept. 3, 2024 from.
- [121] James Prather et al. 2023. The robots are here: navigating the generative AI revolution in computing education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, (Dec. 28, 2023), 108–159. ISBN: 979-8-4007-0405-5. doi:10.1145/3623762.3633499.
- [122] Julian Aron Prenner, Hlib Babii, and Romain Robbes. 2022. Can OpenAI’s codex fix bugs? an evaluation on QuixBugs. In *Proceedings of the Third International Workshop on Automated Program Repair*, 69–75.
- [123] Ben Puryear and Gina Sprint. 2022. Github copilot in the classroom: learning to code with AI assistance. *Journal of Computing Sciences in Colleges*, 38, 1, 37–47. Retrieved Oct. 3, 2023 from <https://www.ccsc.org/publications/journals/NW2022.pdf#page=37>.
- [124] Junaid Qadir. 2023. Engineering education in the era of ChatGPT: promise and pitfalls of generative AI for education. In *2023 IEEE Global Engineering Education Conference (EDUCON)*. 2023 IEEE Global Engineering Education Conference (EDUCON). (May 2023), 1–9. doi:10.1109/EDUCON54358.2023.10125121.

- [125] Yolanda A. Rankin, Jakita O. Thomas, and Sheena Erete. 2021. Black women speak: examining power, privilege, and identity in CS education. *ACM Transactions on Computing Education*, 21, 4, (Oct. 25, 2021), 26:1–26:31. doi:10.1145/3451344.
- [126] Lauren Rich, Heather Perry, and Mark Guzdial. 2004. A CS1 course designed to address interests of women. *ACM SIGCSE Bulletin*, 36, 1, (Mar. 2004), 190–194. doi:10.1145/1028174.971370.
- [127] Mona Rizvi, Thorna Humphries, Debra Major, Meghan Jones, and Heather Lauzun. 2011. A CS0 course using scratch. *Journal of Computing Sciences in Colleges*, 26, 3, 19–27. Retrieved Sept. 25, 2023 from https://www.researchgate.net/profile/Rhys-Price-Jones-2/publication/234790293_Hands-on_education_in_robotics_for_talented_youth_in_developing_countries/links/54218f3a0cf2ce3a91b79201/Hands-on-education-in-robotics-for-talented-youth-in-developing-countries.pdf#page=28.
- [128] Sarah L. Rodriguez and Kathleen Lehman. 2017. Developing the next generation of diverse computer scientists: the need for enhanced, intersectional computing identity theory. *Computer Science Education*, 27, 3, (Oct. 2, 2017), 229–247. _eprint: <https://doi.org/10.1080/08993408.2018.1457899>. doi:10.1080/08993408.2018.1457899.
- [129] Ira J. Roseman. 1996. Appraisal determinants of emotions: constructing a more accurate and comprehensive theory. *Cognition & Emotion*, 10, 3, (May 1996), 241–278. doi:10.1080/026999396380240.
- [130] 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. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. IUI ’23: 28th International Conference on Intelligent User Interfaces. ACM, Sydney NSW Australia, (Mar. 27, 2023), 491–514. ISBN: 979-8-4007-0106-1. doi:10.1145/3581641.3584037.
- [131] James A. Russell. 1980. A circumplex model of affect. *Journal of personality and social psychology*, 39, 6, 1161. Retrieved Nov. 7, 2023 from <https://psycnet.apa.org/journals/psp/39/6/1161/>.
- [132] Caitlin Sadowski and Thomas Zimmermann. 2019. *Rethinking productivity in software engineering*. Springer Nature. Retrieved Sept. 9, 2024 from <https://library.oapen.org/handle/20.500.12657/22839>.
- [133] Norsaremah Salleh, Emilia Mendes, and John Grundy. 2011. Empirical studies of pair programming for CS/SE teaching in higher education: a systematic literature review. *IEEE Transactions on Software Engineering*, 37, 4, (July 2011), 509–525. doi:10.1109/TSE.2010.59.
- [134] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. 2023. Lost at c: a user study on the security implications of large language model code assistants. In 32nd USENIX Security Symposium (USENIX Security 23), 2205–2222. ISBN: 978-1-939133-37-3. Retrieved Mar. 19, 2024 from <https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval>.

- [135] Linda J. Sax, Jennifer M. Blaney, Kathleen J. Lehman, Sarah L. Rodriguez, Kari L. George, and Christina Zavala. 2018. Sense of belonging in computing: the role of introductory courses for women and underrepresented minority students. *Social Sciences*, 7, 8, (Aug. 2018), 122. Number: 8. doi:10.3390/socsci7080122.
- [136] Erik Schluntz. 2025. Raising the bar on SWE-bench verified with claude 3.5 sonnet. Engineering at Anthropic. (Jan. 6, 2025). Retrieved Mar. 21, 2025 from <https://www.anthropic.com/engineering/swe-bench-sonnet>.
- [137] Anshul Shah, Anya Chernova, Elena Tomson, Leo Porter, William G. Griswold, and Adalbert Gerald Soosai Raj. 2025. Students' use of GitHub copilot for working with large code bases. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, (Feb. 18, 2025), 1050–1056. ISBN: 979-8-4007-0531-1. doi:10.1145/3641554.3701800.
- [138] Abdulhadi Shoufan. 2023. Can students without prior knowledge use ChatGPT to answer test questions? an empirical study. *ACM Trans. Comput. Educ.*, 23, 4, (Dec. 11, 2023), 45:1–45:29. doi:10.1145/3628162.
- [139] Phil Steinhorst, Andrew Petersen, and Jan Vahrenhold. 2020. Revisiting self-efficacy in introductory programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. ICER '20: International Computing Education Research Conference. ACM, Virtual Event New Zealand, (Aug. 10, 2020), 158–169. ISBN: 978-1-4503-7092-9. doi:10.1145/3372782.3406281.
- [140] Klaus M. Stiefel and Jay S. Coggan. 2023. The energy challenges of artificial superintelligence. *Frontiers in artificial intelligence*, 6, 1240653. Retrieved Mar. 2, 2026 from <https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2023.1240653/full>.
- [141] John Sweller. 2011. Cognitive load theory. In *Psychology of learning and motivation*. Vol. 55. Elsevier, 37–76. Retrieved Dec. 28, 2023 from <https://www.sciencedirect.com/science/article/pii/B9780123876911000028>.
- [142] Gemini Team. 2023. Gemini: a family of highly capable multimodal models. (Dec. 19, 2023).
- [143] 2014. *The culturally engaging campus environments (CECE) model: a new theory of success among racially diverse college student populations*. Samuel D. Museus. *Higher Education: Handbook of Theory and Research*. Springer Netherlands, Dordrecht, 189–227. ISBN: 978-94-017-8004-9 978-94-017-8005-6. doi:10.1007/978-94-017-8005-6_5.
- [144] Feichin Ted Tschang and Esteve Almirall. 2021. Artificial intelligence as augmenting automation: implications for employment. *Academy of Management Perspectives*, 35, 4, (Nov. 2021), 642–659. doi:10.5465/amp.2019.0062.

- [145] Karthikeyan Umapathy and Albert D. Ritzhaupt. 2017. A meta-analysis of pair-programming in computer programming courses: implications for educational practice. *ACM Transactions on Computing Education (TOCE)*, 17, 4, 1–13.
- [146] UNESCO. 2023. UNESCO survey: less than 10% of schools and universities have formal guidance on AI | UNESCO. UNESCO. (June 7, 2023). Retrieved July 27, 2023 from <https://www.unesco.org/en/articles/unesco-survey-less-10-schools-and-universities-have-formal-guidance-ai>.
- [147] UNESCO. 2025. UNESCO survey: two-thirds of higher education institutions have or are developing guidance on AI use. UNESCO. (Sept. 2, 2025). Retrieved Mar. 2, 2026 from <https://www.unesco.org/en/articles/unesco-survey-two-thirds-higher-education-institutions-have-or-are-developing-guidance-ai-use>.
- [148] Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: integrating LLMs into CS1 instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, (July 3, 2024), 297–303. ISBN: 979-8-4007-0600-4. doi:10.1145/3649217.3653584.
- [149] David Vadas and James R. Curran. 2005. Programming with unrestricted natural language. In *Proceedings of the Australasian Language Technology Workshop 2005*, 191–199. Retrieved May 19, 2025 from <https://aclanthology.org/U05-1027.pdf>.
- [150] 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 (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, (Apr. 28, 2022), 1–7. ISBN: 978-1-4503-9156-6. doi:10.1145/3491101.3519665.
- [151] Marcel Valový and Alena Buchalcevova. 2023. The psychological effects of AI-assisted programming on students and professionals. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME). ISSN: 2576-3148. (Oct. 2023), 385–390. doi:10.1109/ICSME58846.2023.00050.
- [152] Roli Varma. 2010. Why so few women enroll in computing? gender and ethnic differences in students' perception. *Computer Science Education*, 20, 4, (Dec. 2010), 301–316. doi:10.1080/08993408.2010.527697.
- [153] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

- [154] Lev S. Vygotsky. 1978. *Mind in Society: The Development of Higher Psychological Processes*. Michael Cole, Vera John-Steiner, Sylvia Scribner, and Ellen Souberman, (Eds.) Harvard University Press, Cambridge, MA. ISBN: 978-0-674-57629-2.
- [155] Jennifer Wang, Sepehr Hejazi Moghadam, and Juliet Tiffany-Morales. 2017. Social perceptions in computer science and implications for diverse students. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ICER '17: International Computing Education Research Conference. ACM, Tacoma Washington USA, (Aug. 14, 2017), 47–55. ISBN: 978-1-4503-4968-0. doi:10.1145/3105726.3106175.
- [156] Max Weber, Peter R. Baehr, Gordon C. Wells, and Max Weber. 2012. *The Protestant ethic and the "spirit" of capitalism and other writings*. (23. print ed.). *Penguin twentieth-century classics*. Penguin Books, New York. 392 pp. ISBN: 978-0-14-043921-2.
- [157] Justin D. Weisz et al. 2021. Perfection not required? human-AI partnerships in code translation. In MAG ID: 3154248444. (Apr. 14, 2021), 402–412. doi:10.1145/3397481.3450656.
- [158] 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*. SIGCSE 2023: The 54th ACM Technical Symposium on Computer Science Education. ACM, Toronto ON Canada, (Mar. 2, 2023), 172–178. ISBN: 978-1-4503-9431-4. doi:10.1145/3545945.3569830.
- [159] Eric N. Wiebe, Edward Roberts, and Tara S. Behrend. 2010. An examination of two mental workload measurement approaches to understanding multimedia learning. *Computers in Human Behavior*, 26, 3, (May 1, 2010), 474–481. doi:10.1016/j.chb.2009.12.006.
- [160] Terry Winograd. 1972. Understanding natural language. *Cognitive Psychology*, 3, 1, (Jan. 1, 1972), 1–191. doi:10.1016/0010-0285(72)90002-3.
- [161] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE code generation from natural language: promise and challenges. *ACM Transactions on Software Engineering and Methodology*, 31, 2, (Mar. 4, 2022), 29:1–29:47. doi:10.1145/3487569.
- [162] Yuankai Xue, Hanlin Chen, Gina R. Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT help with introductory programming?an experiment of students using ChatGPT in CS1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (ICSE-SEET '24). Association for Computing Machinery, New York, NY, USA, (May 24, 2024), 331–341. ISBN: 979-8-4007-0498-7. doi:10.1145/3639474.3640076.

- [163] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, 62–71.
- [164] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, 4, (Jan. 1, 2023), 100147. doi:10.1016/j.caeai.2023.100147.
- [165] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in computing education: perspectives of students and instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*. 2023 IEEE Frontiers in Education Conference (FIE). (Oct. 2023), 1–9. doi:10.1109/FIE58773.2023.10343467.
- [166] John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, 1050–1055. Retrieved May 19, 2025 from <https://cdn.aaai.org/AAAI/1996/AAAI96-156.pdf>.
- [167] Jingjing Zhao, Tao Song, and Yuxia Sun. 2020. Apihelper: helping junior android programmers learn api usage. *IAENG International Journal of Computer Science*, 47, 1, 92–97. Retrieved Mar. 3, 2026 from https://www.iaeng.org/IJCS/issues_v47/issue_1/IJCS_47_1_12.pdf.

Chapter 2

Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation

Chapter Preamble

As a first step toward Aim 1, this chapter (2) evaluates GitHub Copilot for use by novice programmers using a controlled, quantitative comparison of a solo programming condition to an AI-assisted condition. In addition to objective performance and subjective workload and emotion, we collected a reflective self-grade that revealed the extent to which participants took credit for the performance enhancements furnished by Copilot. The purpose was to reveal any strong harms or benefits of using Copilot that might give initial clues about the appropriateness of allowing novices to use it. These initial findings would help to narrow future lines of questioning away from a strict focus on assisted performance and toward more holistic views of the collaboration experience.

Authorship, Reuse, and Licensing Statement With the explicit consent of coauthors Raymond Pettit  and Sara L. Riggs , this chapter directly reuses contents from [12], a conference proceedings paper led by Nicholas Gardella . The license for this chapter is CC-BY 4.0. ACM allows complete reuse of published papers in doctoral dissertations per the following section of the *ACM Author Rights* policy, retrieved on March 16, 2026: "Authors can include partial or complete papers of their own in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included."

Abstract

Artificial Intelligence-driven Development Environments (AIDEs) offer developers revolutionary computer programming assistance. There is great potential in incorporating AIDEs into computer science education; however, the effects of these tools should be fully examined before doing so. Here, a within-subjects study was conducted to compare the programming performance, workload, emotion, and self-efficacy of seventeen novices coding with and without use of the GitHub Copilot AIDE under time pressure. Results showed that using the AIDE significantly increased programming efficiency and reduced effort and mental workload but did not significantly impact emotion or self-efficacy. However, participants' performance improved with more experience using the AI, and their self-efficacy followed. The results suggest that students who try AIDEs will likely be tempted to use them for time-sensitive work. There is no evidence that providing AIDEs will aid struggling students, but there is a clear need for students to practice with AI to become competent and confident using it.

2.1 Introduction

Large Language Models are powering a wave of increasingly capable AI-based code generators. One of the most popular and accessible tools employing generative AI technology for programming is GitHub Copilot. This AIDE was released publicly in June 2022 and has been coined as “your AI pair programmer.” Copilot can write multiline, multilingual solutions, match context, write tests, create documentation, and explain code in natural language [15]. Currently, verified students can access Copilot for free.

AIDEs are highly capable in solving simple tasks that novices tend to encounter as they are developing computer programming competencies [8, 11]. This raises concerns that students will overuse AIDEs, fail to learn from educational programming assignments, and be under-prepared for their careers [5]. Leaders in education are calling for policies to address these potential risks [14].

Kazemitabaar et al. found that an AIDE could improve young novices' performance on self-paced programming practice problems without leading to lower performance on a learning retention test [21]. However, to our knowledge, no studies have investigated the impact of AIDEs on novice programmers under time pressure. Furthermore, while researchers tend to show that AIDEs are beneficial to programmer well-being [21, 35, 37], there have been limited studies investigating how AI impacts novice programmers' workload, emotion, or self-efficacy.

Performance, workload, and emotion all have important ties to self-efficacy and may reveal how AI can be used to help students persist in computer science education. The following research questions are investigated in this study:

- **RQ1:** How do AIDEs affect the performance, workload, emotion, and self-efficacy of novice programmers under time pressure?
- **RQ2:** How do AIDEs influence the effects of additional time spent programming on novices' performance and self-efficacy?

2.2 Related Work

2.2.1 Human-AIDE Interaction

While many studies have evaluated AIDEs in isolation (e.g., [8, 26, 39]), it is more useful to consider how people use them. One study recruited 31 participants to code with and without an AIDE. While participants largely reported neutral or positive experiences, they did not show any significant improvements in task correctness or completion times [37]. Another study investigated 21 students working with and without GitHub Copilot. The authors concluded that code written with Copilot was comparable in complexity to code written without it, though static code analysis showed a significantly higher Halstead Difficulty score (a measure of the difficulty of code to read and write). Eye tracking metrics on 15 participants' data also showed significantly shorter and fewer code token fixations when using Copilot [1]. This is especially concerning given AIDE suggestions have been shown to introduce security vulnerabilities into code [29]. Finally, a study

of the performance and experience of 24 intermediate and advanced programmers working with and without GitHub Copilot also found no significant performance improvements. Though, participants overwhelmingly preferred Copilot to standard autocomplete [35]. Together, these studies show that the costs and benefits from using AIDEs are varied and not well defined. Further, there are potential risks posed by over-reliance on the AI.

2.2.2 Performance of Novices Using AIDEs

Researchers at University of Auckland used an AI code generator on the assessments in their introductory computer science (CS1) course. It performed well enough to earn exam scores that placed it above 75% of the students in the course [11]. This study raised alarms about how over-reliance on AI-based programming tools might harm novice programmers' learning. Popovici reported a similar finding for assignments in a functional programming course [31].

Related work has begun to study novices using AIDEs directly. Kazemitabaar et al. used a custom AIDE in a study of 69 students between the ages of 10-17 [21]. The students who learned with AI performed significantly better on their practice assignments, and there was no evidence of reduced learning retention on summative exams, on which they worked unassisted. The AI-assisted students who performed best on their entrance exam also showed significant improvements in learning based on the summative exams, indicating that AIDEs can potentially provide learning benefits to more advanced learners [21]. Given these results contrast with the findings of [35, 37], it is important to conduct a similar study in a more controlled scenario—the goal of the work here.

2.2.3 Measuring Well-being

In addition to performance, the well-being of the programmer is also important to consider. As a challenging task, programming imparts a “human cost” known broadly as “workload” [16]. The current standard for measuring workload in human factors research is the NASA Task Load Index (TLX). It contains six dimensions: mental, physical, temporal, performance, effort, and frustra-

tion. While workload is not strictly an aspect of well-being, it can have major consequences on well-being in the long run. Of the six dimensions, mental and effort tend to be highest during educational programming, while physical demand tends to be lowest [2]. In the case of [13], increased learning performance was associated with lower effort and “task demand,” a combination of the mental and physical dimensions together.

IEEE 7010-2020 is an active standard for assessing the benefits and harms of autonomous and intelligent systems on human well-being [19]. It discusses “indicators” for assessing well-being within 12 “domains.” One of these domains refers to individual feelings and emotions, indicated by subjective reports of positive (“happy, calm, peaceful, etc.”) and negative affects (“sad, depressed, anxious, etc.”). Russel’s affect model, which we employ, augments a valence dimension with an arousal dimension to account for a wider range of moods [32].

The IEEE standard [19] also defines a domain of psychological well-being. This refers to how one feels about what they are doing day-to-day (as opposed to their overall life satisfaction), indicated by subjective reports such as purpose and self-efficacy. Self-efficacy—i.e., one’s own perception of their ability to succeed in the future—can change based on how someone thinks they performed during past experiences. Therefore, we employ a simple letter grade for participants to assess their own performance.

2.2.4 Effects of Well-being in CS Education

Due to lack of skills and experience, programming tasks can be very challenging for novices [25]. Mismatches between students’ abilities and the demands of a CS1 course can lead them to drop the course [3, 30] or the CS major altogether [6]. According to Cognitive Load Theory, excessive load, which TLX can measure [13, 34], can be harmful to students’ learning. The Zone of Proximal Flow Model considers a similar phenomenon, warning that excessively easy coursework can also harm learning via boredom [4]. The findings of [7] validate the harms of both boredom and frustration in the learning process, and together, these examples demonstrate the significance of optimizing workload for novice programmers.

Self-efficacy is another key predictor for academic success [17, 36]. It is also an outcome of the learning process and correlates with positive emotions [24]. Programmers' emotions are often overlooked, but can have significant impacts on performance [10] and student experience [22]. Therefore, it is critical that educational programming experiences help to foster both positive emotions and strong self-efficacy in students.

2.3 Method

2.3.1 Participants

IRB approval was obtained, and seventeen undergraduate participants (11 Female, 5 Male, 1 Undisclosed) were recruited from a 1000-level Introduction to Programming (CS1) course. While the majority were pursuing STEM majors, four were pursuing a non-STEM degree. All students in the course were invited to participate; however, given that recruitment required several steps, the sample may have been biased toward students with strong organizational skills. All participants self-identified as novice programmers at the start of their course, but six identified as intermediate on the day of participation; none identified as advanced. Participants were compensated with \$15 per hour in gift cards. Two additional \$10 gift cards were awarded to the top two scoring participants to encourage them to give their best effort and to mitigate the impact of the novelty effect on use of the AIDE.

2.3.2 Experimental Setup

Participants worked on a 21" iMac in the Visual Studio Code (VS Code) integrated development environment with a standard keyboard and mouse. The official GitHub Copilot extension was installed alongside a Python 3 environment. A custom interface allowed participants to access and submit tasks. They could see the time remaining in the trial, a PDF document describing the task, and a Python editor with a skeleton function for them to implement. Participants could run and test their code using either buttons in the IDE or with commands in the terminal. If their

solution for a task passed all the tests provided in the dataset (see 2.3.3), a button would allow them to advance to the next task in the trial.

2.3.3 Tasks

Task Sourcing

Tasks were sourced from the 164-task, human-written HumanEval Dataset [8]. These tasks are a common tool for evaluating the coding performance of LLMs [26, 27, 39]. Due to the age of the dataset and its role as a benchmark, there is a possibility that these tasks have appeared in training or evaluation of newer deployments of GitHub Copilot, leading to unusually high performance on the tasks. Therefore, the tasks chosen gave the participants a high ceiling on their success, but only to the extent they could effectively utilize the AI.

Task Preparation

To further address this concern, the context made available to Copilot by default was very minimal, including only the name of the function and its arguments. This forced participants to introduce context to Copilot with their actions and entries, rather than having it immediately offer a solution. At the time of study, GitHub Copilot only had access to text files that were actively open. The testing code file was closed and hidden by default but could be opened. In their original form, tasks contained comments with the task prompt. However, these prompt comments were removed from the task files and moved to a PDF file. This strategy encoded the text graphically such that the participant could read and even copy-paste the prompt, but GitHub Copilot could not take and account for the prompt on its own. Participants were neither encouraged nor prohibited to use any strategy to provide context to Copilot.

Task Selection

Nine teaching staff from the Introduction to Programming class assisted the research team in selecting and arranging four pools of similar programming tasks from HumanEval. In total, 32

tasks were selected (eight tasks for each of four pools) and reviewed by at least one rater. Raters timed their own completion of the task and estimated its feasibility and subjective difficulty for their students, noting any special knowledge that might be required (such as a geometry formula or name of a required library). Based on these ratings, the research team organized tasks into six roughly similar difficulty levels. Then, tasks were distributed evenly by difficulty between the pools. The tasks within each pool were ordered to have increasing difficulty, with two easier tasks at the end. The goal of this approach was to avoid participants becoming stuck on a difficult task early on. For example, an easy task (id: 35) was placed first in pool D and asked to find the maximum element in a list of numbers. A harder task (id: 6) was placed fifth in pool C and asked for the maximum depth of a string of nested parentheses. The early tasks were also focused toward topics that had been covered in Introduction to Programming and did not require special knowledge, which was an issue in [35]. At the end, easier tasks requiring special knowledge (e.g., a less common standard library) were included for any participants who were able to pass even the highly difficult tasks. The HumanEval tasks used were: Pool A (24, 3, 0, 12, 26, 37, 51, 49), Pool B (53, 52, 18, 69, 25, 70, 7, 2), Pool C (23, 74, 14, 9, 6, 161, 19, 22), and Pool D (35, 67, 72, 5, 13, 38, 20, 45).

2.3.4 Experimental Design

General Design

The study employed a within-subjects design using within-subjects factors of AIDE availability (solo v. with AI) and experience (first solo/AI trial v. second solo/AI trial). Each participant completed two trials with the AIDE and two trials without (aside from one participant who completed only one trial without). The order of task pools, order of treatments, and pairings of task pool to treatment were counterbalanced across participants. However, as mentioned previously, the order of tasks within each task pool was fixed.

No Internet Access

Standard Python IntelliSense was available during control trials to provide an autocomplete functionality like that of the PyCharm editor participants used in their course. After consideration, use of internet search or outside resources was not permitted under either condition. The purpose of this design was to compare AI-assisted programming to solo programming, rather than to internet-assisted programming. Participants were also expected to be highly experienced using internet search. On the other hand, only 2 of the 17 participants had any prior experience using AIDEs. Moreover, we were concerned that the internet might expose verbatim solutions, e.g. Hugging Face, which hosts a directly viewable table of all solutions. Previous studies either allowed internet access [35, 37] or did not specify [1, 21], so this choice is important to consider.

2.3.5 Procedure

After electronic informed consent was obtained, the participant was provided an overview of the study and allowed to ask questions. Next, a short background questionnaire was administered covering identity characteristics and prior experience. We created several training videos, which participants watched before completing relevant practice drills. These covered the process for navigating tasks, running unit tests, submitting tasks, and using the AIDE. One video covered the risks and benefits of the AIDE, emphasizing that while it can recognize patterns and context, it can also propagate errors in code or produce faulty outputs. Participants were allowed to practice with the AIDE, testing facilities, and submission controls for as long as they needed to become comfortable before moving on. Following training, participants completed their task pools in their unique assigned order along with emotion and task reflection questionnaires according to the timeline in Figure 2.1.

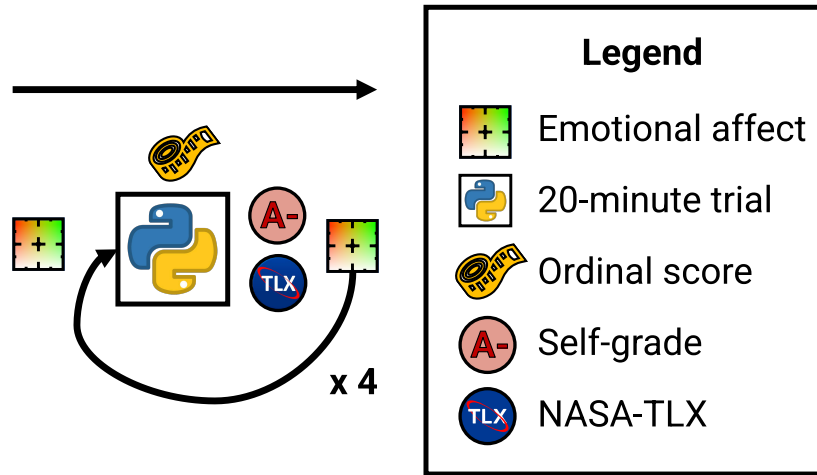


Figure 2.1: Experimental timeline for each trial that was repeated four times (x4).

2.3.6 Data Collection

Emotion

Participants completed an emotional affect questionnaire before and after each trial (20 minutes to work through a programming task pool sequentially). A before-to-after measurement technique was chosen to help isolate emotional impacts of each trial from other mood factors. Therefore, the measure for each trial was the change in rating (after – before).

Affect questions were based on Russell’s [32] circumplex model of affect. Participants estimated their emotional valence (see Figure 2.2, top), followed by their emotional arousal (see Figure 2.2, right), both on seven-point Likert scales (coded 1-7). Both questions were accompanied by a Valence v. Arousal visual aid (see Figure 2.2, bottom left), which was thoroughly explained to participants and visually aligned with the presentation of Likert choice bubbles.

Performance

Participants worked for 20 minutes on each assigned task pool either with or without AI assistance. They submitted tasks using the VS Code interface (see 2.3.2), which used PyTest to validate the performance of their solution against the unit tests provided in the HumanEval dataset.

If their code passed all these tests, they were allowed to advance to the next task in the pool. Their performance on each of the four trials was measured using an ordinal performance score as the tasks were not of uniform difficulty.

Workload

Following each trial, a digital rendition of the widely used NASA Task Load Index (NASA-TLX) was administered to measure six dimensions of workload [16]. Given the TLX is a reflective questionnaire, it was administered only after each programming trial. Special effort was made to explain the TLX performance scale, which has a counterintuitive layout participants might otherwise overlook.

Self-Efficacy

Self-efficacy refers generally to one's perception of their ability to succeed at something (in the future). Our measure for this idea was instead reflective, asking about how a participant thought they performed on the previous trial with the critical assumption that recognizing past success can translate to self-efficacy on similar tasks in the future. After completing a trial, participants were asked to grade their own work on that trial using a 13-point letter grade scale (F, D- ... A+). In the results, these values are coded 1-13, so a change in self-grade of one corresponds to an increase by one half letter grade (e.g. C+ to B-) and a self-grade of eight, for example, corresponds to a B-.

2.3.7 Data Analysis

Statistical testing was performed on paired measurements using the within-subjects factors of AIDE availability and experience. Due to the small sample size and presence of ordinal data, the critical assumptions for each test were validated prior to performing the test. A paired t-test was used if the paired differences were at least interval-level and normally distributed (as assessed by the Shapiro-Wilk normality test) with no extreme outliers (using the criteria in [20]) [33]. If

Valence: Which best describes your emotional state?

	Extremely Negative	Negative	Slightly Negative	Neutral	Slightly Positive	Positive	Extremely Positive
Valence	☐	☐	☐	☐	☐	☐	☐

Upset

↑

Arousal

↓

←

Valence

→

↑

Arousal

↓

Bored

←

Valence

→

Calm

Arousal: Which best describes your alertness and intensity of <Valence Choice> feelings?

Extremely High	☐
High	☐
Slightly High	☐
Normal	☐
Slightly Low	☐
Low	☐
Extremely Low	☐

Figure 2.2: Exact questions asked to participants about their emotional affect alongside a complimentary visualization.

Note. The visualization is based on the Russell [32] circumplex of affect model.

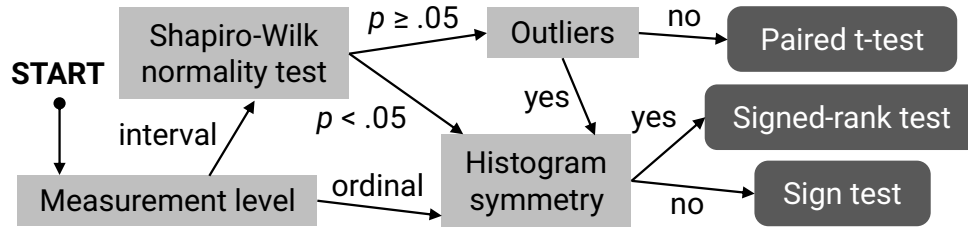


Figure 2.3: Decision tree for choosing statistical tests.

any of these assumptions was not met, but the paired differences were symmetrical about some midline, the Wilcoxon signed rank test was used [33]. If asymmetry was clearly apparent, the sign test was used via a normal approximation of the binomial distribution [33, 23]. Figure 2.3 shows the decision tree used to choose a test for each comparison in the results.

When examining the AIDE availability factor, paired data were produced by aggregating participants' trials by condition. Since at most two values were being aggregated at once, the mean and median were equivalent in every case. NASA-TLX ratings were rounded to the nearest 5 points (100-point scale) to mimic the resolution of the paper-and-pencil version [16].

All statistics were computed in R using two-sided hypothesis tests. Paired t-tests were calculated with `t.test(paired=T)` from the standard `stats` library. Wilcoxon signed-rank tests were performed using `wilcoxsign_test(distribution="exact")` from the `coin` [18] R language package. The sign test was calculated by normal approximation using the Yates [38] continuity corrected equal proportions test with `prop.test(p=0.5,correct=T)` from the standard `stats` library.

For measures where t-tests were appropriate, the data are presented as means ($\bar{x}$); otherwise, the data are presented as medians ($\tilde{x}$). The change shown is the mean or median of the paired differences. Effect sizes were computed using Cohen's [9] d for all statistical tests. These values should be compared with caution since the hypotheses tested differ among the three tests. To make the calculation as consistent as possible, the t or z statistic was divided by the square root of total sample size, counting all replicates used in a test to avoid inflating the effect size [28]. The significance threshold was set at $\alpha=.05$ for all statistical tests.

2.4 Results

2.4.1 Effects of AI Availability (RQ1)

Descriptive Statistics

Trends (Table 2.1) were toward improved performance with the AI, a more positive emotional valence effect, a more arousing emotional arousal effect, decreases in most dimensions of NASA-TLX workload, and increases in self-grade (see Section 2.3.6 for letter to numerical grade conversion).

Table 2.1: Descriptive statistics for changes in measures from solo trials to trials with the AIDE.

	Solo	with AI	Change
Score	$\tilde{x} = 2$	$\tilde{x} = 2.5$	$\widetilde{\Delta x} = 1$
Δ Valence	$\bar{x} = -0.088$	$\bar{x} = 0.118$	$\overline{\Delta x} = 0.206$
Δ Arousal	$\tilde{x} = 0$	$\tilde{x} = 0.5$	$\widetilde{\Delta x} = 0.5$
TLX Mental	$\bar{x} = 58.382$	$\bar{x} = 49.412$	$\overline{\Delta x} = -8.971$
TLX Physical	$\tilde{x} = 5$	$\tilde{x} = 2.5$	$\widetilde{\Delta x} = 0$
TLX Temporal	$\bar{x} = 54.559$	$\bar{x} = 49.265$	$\overline{\Delta x} = -5.294$
TLX Performance	$\bar{x} = 61.912$	$\bar{x} = 55.882$	$\overline{\Delta x} = -6.029$
TLX Effort	$\tilde{x} = 62.5$	$\tilde{x} = 55$	$\widetilde{\Delta x} = -5$
TLX Frustration	$\bar{x} = 47.059$	$\bar{x} = 44.853$	$\overline{\Delta x} = -2.206$
Self-Grade	$\bar{x} = 7.647$	$\bar{x} = 7.912$	$\overline{\Delta x} = 0.265$

Statistical Tests

Tests were performed according to the prescribed method. The results can be found in Table 2.2. The Wilcoxon signed rank test indicated a statistically significant median increase in performance and a statistically significant median decrease in the NASA-TLX effort dimension with the AIDE. The paired t-test indicated a statistically significant mean decrease in the NASA-TLX mental demand dimension. None of the other results was significant.

Table 2.2: Statistical comparisons between solo and with AI trials for each measure.

	Statistic	<i>p</i>	<i>d</i>
Score	$Z_{\text{Wilcoxon}} = 3.012$	.001*	0.368
Δ Valence	$t(16) = 0.800$	.436	0.098
Δ Arousal	$Z_{\text{Wilcoxon}} = 1.329$	.191	0.162
TLX Mental	$t(16) = -2.487$	.024*	-0.304
TLX Physical	$Z_{\text{Sign Test}} = -0.289$	.773	-0.035
TLX Temporal	$t(16) = -1.338$	.200	-0.163
TLX Performance	$t(16) = -1.248$	.230	-0.152
TLX Effort	$Z_{\text{Wilcoxon}} = -2.023$	.043*	-0.247
TLX Frustration	$t(16) = -0.563$	.581	-0.069
Self-Grade	$t(16) = 0.765$	.455	0.093

An asterisk (*) denotes significance.

2.4.2 Effects of Experience (RQ2)

Descriptive Statistics

Trends (Table 2.3) were not apparent between solo trials, but actual (Score) and perceived (Self-Grade) performance both trended higher on the second AI trial compared to the first.

Table 2.3: Descriptive statistics comparing the participants' first and second trial scores and self-grades for the solo and with AI conditions.

		1st Trial	2nd Trial	Change
Solo	Score	$\bar{x} = 2$	$\bar{x} = 2$	$\widetilde{\Delta x} = 0$
	Self-Grade	$\bar{x} = 8$	$\bar{x} = 7.25$	$\widetilde{\Delta x} = 0$
with AI	Score	$\bar{x} = 2$	$\bar{x} = 3$	$\widetilde{\Delta x} = 1$
	Self-Grade	$\bar{x} = 7.118$	$\bar{x} = 8.706$	$\overline{\Delta x} = 1.588$

Statistical Tests

Tests were performed according to the prescribed method. The results can be found in Table 2.4. The Wilcoxon signed rank test indicated a statistically significant median increase in performance

on the second trial with AI. The paired t-test indicated a statistically significant mean increase in the letter grade the participant assigned to their own work on the second trial with AI. That is, participants perceived that they performed better after working with the AI for the second time, and, in fact, they did. Neither actual nor perceived performance changed significantly from the first *solo* trial to the second, however.

Table 2.4: Statistical comparisons between the first and second trial scores and self-grades for the solo and with AI conditions.

		Statistic	<i>p</i>	<i>d</i>
Solo	Score	$Z_{\text{Wilcoxon}} = 0.090$	1.000	0.016
	Self-Grade	$Z_{\text{Wilcoxon}} = -0.517$	.594	-0.091
with AI	Score	$Z_{\text{Wilcoxon}} = 2.029$	.046*	0.348
	Self-Grade	$t(16) = 2.567$	.021 *	0.440

An asterisk (*) denotes significance.

2.5 Discussion

2.5.1 RQ1: How do AIDEs affect the performance, workload, emotion, and self-efficacy of novice programmers under time pressure?

The increase in performance is consistent with the findings of [21], where participants worked on self-paced, skippable tasks with feedback from human graders. Here, participants were required to complete tasks in order under time pressure with only standard, automated feedback. Even with bonus compensation at stake, participants used the AIDE enough to be more productive than working without it. This result differs from the findings of [35, 37], where participants could use the internet in the control condition and saw no performance benefit from the AIDE. Since it is not clear that participants in [21] were expressly permitted to use the internet in the control condition, this may explain why [35, 37] did not see a performance improvement.

As in [2], we observed very low physical workload generally, with medians below 10 on the 100-point scale, compared to over 40 for the other dimensions. The observation in [2] that mental

demand and effort respond most when students program may explain the significant reductions that we observed from use of the AI. However, the performance dimension, which is more like the self-grading metric than the other workload dimensions, was particularly high in both conditions despite neither reaching statistical significance nor standing out in [2]. This discrepancy may be attributable to the counterintuitive layout of this scale or the fact that we used raw TLX ratings, while [2] reported ratings scaled according to contribution to overall workload. Our findings also indicate a possible learning benefit for students using AI, given that [13] found significant mental demand and effort decreases in the condition where students learned significantly more.

As time pressure was identical both with and without AI, it is unsurprising that temporal demand was comparable. More notable is that frustration, emotional effects, and self-grade were comparable, too. The limited sample size may be a contributor to the lack of significance seen here. In line with workload decreases, the emotional arousal data showed a non-significant trend for programming with AI to have a more arousing effect compared to solo programming. Valence change, frustration, and self-grade also showed that using AI trended in beneficial directions but were not significant. The non-significance of the self-grade increase is best interpreted as participants attributing the improvement to the AI rather than to themselves.

2.5.2 RQ2: How do AIDEs influence the effects of additional time spent programming on novices' performance and self-efficacy?

When considering the trials with AI and the solo trials separately, distinct experience/learning effects appeared. Participants' actual and perceived performance (self-grade) increased significantly between their first trial with AI to their second trial with AI. Previous studies either did not report such learning effects [35, 37], or did not use the necessary study design [21]. For solo trials, however, neither actual nor perceived performance changed significantly. This highlights a difference in growth potential and stresses the importance of practice for novices programming with AI. Secondly, because experience increased participants' self-grade only on AI trials, AI-specific programming techniques may provide a new avenue for novices to build self-efficacy.

That is, honing AI-collaboration skills (e.g. prompting) helps novices build confidence that may translate into academic success later on [17, 36].

2.6 Limitations

While participants had access to autocomplete in the control condition, they did not have other resources such as notes or internet search. This choice was deliberate (see Section 2.3.4 for rationale), but it presents a threat to ecological validity. Future studies might consider other tasks that cannot be found online so that a more realistic comparison can be made. The small sample size and lack of diversity present threats to external validity. Participants were recruited at a selective R1 institution in the United States; the findings may not generalize to other student populations or other countries.

2.7 Conclusion and Future Work

In this study, novice programmers wrote Python code with and without AI assistance from the GitHub Copilot AIDE. When working with the AIDE, participants obtained significant benefits to their programming performance, mental workload, and effort. There was no evidence to support any differences in emotion or self-efficacy between programming alone and with AI. However, additional experience programming with the AI yielded significant increases in both actual and perceived performance that were not matched in the solo condition. Educators should be advised that AIDEs will provide tempting benefits for students working on assignments under time pressure. However, they should also consider the importance of experiential learning with AIDEs and their potential to help struggling students build self-efficacy with programming via new skillsets.

Future work should attempt to extend these findings with broader groups of participants and with comparisons to internet-assisted programming or human-to-human pair programming. Ideally, future studies can also evaluate novices using AIDEs over longer periods of time and with deeper focus on learning-specific outcomes. Since performance with the AI increased due to

experience, it may be worth exploring the amount of time needed using AIDEs for students to reach a performance plateau. Finally, as AI becomes increasingly accessible and useful, educators should debate about the extent to which solo programming abilities should be expected of graduates moving forward.

Acknowledgments

This work was supported by the UVA Distinguished Fellowship, NSF NRT (1829004) & GRFP (1842490), and the Virginia CCI CVN (GR102900). We thank Dylan Do, Zack Landsman, Michael Lin, Alex Myers, Eric Nguyen, Jacob Rice, Allison Sawyer, Olivia Seto, and Alice Warner for assistance in selecting and arranging HumanEval tasks.

References

- [1] Naser Al Madi. 2023. How readable is model-generated code? examining readability and visual inspection of GitHub copilot. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. Association for Computing Machinery, New York, NY, USA, (Jan. 5, 2023), 1–5. ISBN: 978-1-4503-9475-8. doi:10.1145/3551349.3560438.
- [2] Naser Al Madi, Siyuan Peng, and Tamsin Rogers. 2022. Assessing workload perception in introductory computer science projects using NASA-TLX. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education. SIGCSE 2022: The 53rd ACM Technical Symposium on Computer Science Education*. ACM, Providence RI USA, (Feb. 22, 2022), 668–674. ISBN: 978-1-4503-9070-5. doi:10.1145/3478431.3499406.
- [3] Lecia J. Barker, Charlie McDowell, and Kimberly Kalahar. 2009. Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin*, 41, 1, (Mar. 4, 2009), 153–157. doi:10.1145/1539024.1508923.
- [4] Ashok R. Basawapatna, Alexander Repenning, Kyu Han Koh, and Hilarie Nickerson. 2013. The zones of proximal flow: guiding students through a space of computational thinking skills and challenges. In *Proceedings of the ninth annual international ACM conference on International computing education*

- research. ICER '13: International Computing Education Research Conference. ACM, San Diego San California USA, (Aug. 12, 2013), 67–74. ISBN: 978-1-4503-2243-0. doi:10.1145/2493394.2493404.
- [5] 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*, 500–506.
- [6] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. 2008. Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. *ACM SIGCSE Bulletin*, 40, 1, (Mar. 12, 2008), 402–406. doi:10.1145/1352322.1352274.
- [7] Nigel Bosch and Sidney D'Mello. 2017. The affective experience of novice computer programmers. *International Journal of Artificial Intelligence in Education*, 27, 1, (Mar. 1, 2017), 181–206. doi:10.1007/s40593-015-0069-5.
- [8] Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- [9] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Academic press.
- [10] Nasrin Dehbozorgi, Mary Lou Maher, and Mohsen Dorodchi. 2020. Sentiment analysis on conversations in collaborative active learning as an early predictor of performance. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9. Retrieved Nov. 30, 2023 from <https://ieeexplore.ieee.org/abstract/document/9274119/>.
- [11] 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 (ACE '22)*. Association for Computing Machinery, New York, NY, USA, (Feb. 14, 2022), 10–19. ISBN: 978-1-4503-9643-1. doi:10.1145/3511861.3511863.
- [12] Nicholas Gardella, Raymond Pettit, and Sara L. Riggs. 2024. Performance, workload, emotion, and self-efficacy of novice programmers using AI code generation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2024: Innovation and Technology in Computer Science Education. ACM, Milan Italy, (July 3, 2024), 290–296. ISBN: 979-8-4007-0600-4. doi:10.1145/3649217.3653615.
- [13] Peter Gerjets, Katharina Scheiter, and Richard Catrambone. 2006. Can learning from molar and modular worked examples be enhanced by providing instructional explanations and prompting self-explanations? *Learning and Instruction*, 16, 2, 104–121. Retrieved Dec. 28, 2023 from <https://www.sciencedirect.com/science/article/pii/S0959475206000144>.

- [14] Stefania Giannini. 2023. Reflections on generative AI and the future of education. (2023). Retrieved July 27, 2023 from <https://unesdoc.unesco.org/ark:/48223/pf0000385877>.
- [15] GitHub. 2023. GitHub copilot · your AI pair programmer. GitHub. Retrieved July 24, 2023 from <https://github.com/features/copilot>.
- [16] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (task load index): results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.
- [17] Toni Honicke and Jaclyn Broadbent. 2016. The influence of academic self-efficacy on academic performance: a systematic review. *Educational research review*, 17, 63–84. Retrieved Dec. 29, 2023 from <https://www.sciencedirect.com/science/article/pii/S1747938X15000639>.
- [18] Torsten Hothorn, Kurt Hornik, Mark A. van de Wiel, Henric Winell, and Achim Zeileis. 2015. Conditional inference procedures in a permutation test framework. *Coin package (R version 3.1. 3)*. Retrieved April, 6, 2016.
- [19] IEEE. 2020. IEEE recommended practice for assessing the impact of autonomous and intelligent systems on human well-being. *IEEE Std 7010-2020*, (May 2020), 1–96. Conference Name: IEEE Std 7010-2020. doi:10.1109/IEEESTD.2020.9084219.
- [20] Boris Iglewicz and David C. Hoaglin. 1993. *Volume 16: how to detect and handle outliers*. Quality Press. Retrieved Nov. 30, 2023 from.
- [21] 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, (Apr. 19, 2023), 1–23. ISBN: 978-1-4503-9421-5. doi:10.1145/3544548.3580919.
- [22] Paivi Kinnunen and Beth Simon. 2010. Experiencing programming assignments in CS1: the emotional toll. In *Proceedings of the Sixth international workshop on Computing education research*. ICER '10: International Computing Education Research Workshop. ACM, Aarhus Denmark, (Aug. 9, 2010), 77–86. ISBN: 978-1-4503-0257-9. doi:10.1145/1839594.1839609.
- [23] Mike LePine. 2022. Chapter 9.4: distribution needed for hypothesis testing. In *College Statistics*. Book Title: College Statistics. St. Clair College AA&T, (July 1, 2022). Retrieved Dec. 1, 2023 from <https://ecampusontario.pressbooks.pub/sccstatistics/chapter/distribution-needed-for-hypothesis-testing/>.

- [24] Alex Lishinski and Joshua Rosenberg. 2021. All the pieces matter: the relationship of momentary self-efficacy and affective experiences with CS1 achievement and interest in computing. In *Proceedings of the 17th ACM Conference on International Computing Education Research*. ICER 2021: ACM Conference on International Computing Education Research. ACM, Virtual Event USA, (Aug. 16, 2021), 252–265. ISBN: 978-1-4503-8326-4. doi:10.1145/3446871.3469740.
- [25] Michael McCracken et al. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working group reports from ITiCSE on Innovation and technology in computer science education* (ITiCSE-WGR '01). Association for Computing Machinery, New York, NY, USA, (Dec. 1, 2001), 125–180. ISBN: 978-1-4503-7359-3. doi:10.1145/572133.572137.
- [26] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories* (MSR '22). Association for Computing Machinery, New York, NY, USA, (Oct. 17, 2022), 1–5. ISBN: 978-1-4503-9303-4. doi:10.1145/3524842.3528470.
- [27] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: an open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.
- [28] Julie Pallant. 2007. *SPSS Survival Manual*. Vol. 14. Open University Press New York, NY, USA. ISBN: 978 0335223664.
- [29] 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.
- [30] Andrew Petersen, Michelle Craig, Jennifer Campbell, and Anya Tafliovich. 2016. Revisiting why students drop CS1. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli Calling 2016: 16th Koli Calling International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 24, 2016), 71–80. ISBN: 978-1-4503-4770-9. doi:10.1145/2999541.2999552.
- [31] Matei-Dan Popovici. 2023. ChatGPT in the classroom. exploring its potential and limitations in a functional programming course. *INTERNATIONAL JOURNAL OF HUMAN-COMPUTER INTERACTION*, (Oct. 17, 2023). Num Pages: 12 Place: Philadelphia Web of Science ID: WOS:001085897000001. doi:10.1080/10447318.2023.2269006.
- [32] James A. Russell. 1980. A circumplex model of affect. *Journal of personality and social psychology*, 39, 6, 1161. Retrieved Nov. 7, 2023 from <https://psycnet.apa.org/journals/psp/39/6/1161/>.

- [33] Laerd Statistics. 2015. *Tutorials for SPSS Statistics. Statistical tutorials and software guides*. Retrieved Dec. 12, 2023 from <https://statistics.laerd.com/>.
- [34] John Sweller. 2011. Cognitive load theory. In *Psychology of learning and motivation*. Vol. 55. Elsevier, 37–76. Retrieved Dec. 28, 2023 from <https://www.sciencedirect.com/science/article/pii/B9780123876911000028>.
- [35] 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 (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, (Apr. 28, 2022), 1–7. ISBN: 978-1-4503-9156-6. doi:10.1145/3491101.3519665.
- [36] Kimberly Wilson and Anupama Narayan. 2016. Relationships among individual task self-efficacy, self-regulated learning strategy use and academic performance in a computer-supported collaborative learning environment. *Educational Psychology*, 36, 2, (Feb. 7, 2016), 236–253. doi:10.1080/01443410.2014.926312.
- [37] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE code generation from natural language: promise and challenges. *ACM Transactions on Software Engineering and Methodology*, 31, 2, (Mar. 4, 2022), 29:1–29:47. doi:10.1145/3487569.
- [38] Frank Yates. 1934. Contingency tables involving small numbers and the χ^2 test. *Supplement to the Journal of the Royal Statistical Society*, 1, 2, 217–235. Retrieved Dec. 12, 2023 from https://www.jstor.org/stable/2983604?casa_token=5sU4ElyzangAAAAA:4L2riIvZWByb2g4dXqZaPdZm6I2L97AJVKmeFdFPoESfurrgguVogadVePgFdc5r8ajaroXziWoqrkuzDcxMY1JH8zDzcvm-qZ83DRXG60xphsldoqTA.
- [39] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, 62–71.

Chapter 3

Novice Interactions with GitHub Copilot Under Time Pressure: A Deductive Qualitative Interview Study

Chapter Preamble

After evaluating GitHub Copilot's quantitative effects in Chapter 2, this chapter (3) furthers Aim 1 through a qualitative exploration of human-AIDE interaction. Through a human-computer interaction lens, this work describes the nature of the interaction and its impacts on the human programmer through a set of interviews. The qualitative analysis supports a number of studies with similar findings, adding additional focus on novice programmers and time-pressured task scenarios through interview data collected after the programming experiment described in Chapter 2. The results of this study provide context to describe the *what* and the *why* of human-AIDE interaction; participants' comments allude to unique hazards for novices during time-pressured, AI-assisted programming that were not revealed by the quantitative findings from Chapter 2.

Authorship, Reuse, and Licensing Statement This chapter was coauthored with Sara L. Riggs  and explores a dataset that overlaps with Chapter 6, but Chapters 3 and 6 explore different research questions, and Chapter 6 analyzes additional data. For additional information, see the Authorship, Reuse, and Licensing Statement in Chapter 6. All coauthors across both chapters (Sara L. Riggs , Claire Wood , and Ellis Barnes ) provided explicit consent for the inclusion of this chapter in this dissertation. The content from this chapter may appear in a future publication. The license for that paper applies to this chapter. The fallback license is listed on the copyright page for this dissertation.

Seeing a full, detailed, qualitative picture of this research area provides an integral foundation for informing policy based on concrete human experiences. This chapter briefly touches on the concept of *trust* as it relates to human-AIDE interaction, but there was not sufficient available data to explain trust dynamics in-depth. The next chapter (4) addresses this challenge, concluding Aim 1 with a quantitative analysis of human-AIDE trust dynamics.

Abstract

Adapting computer science education to the rapid advancement of Artificial Intelligence requires a thorough understanding of how novice programmers interact with and perceive Artificial Intelligence-driven Development Environments (AIDEs). Numerous qualitative studies have investigated human-AIDE interaction, but there have been limited attempts to unify this work within a single framework. Here, we reviewed 11 qualitative studies of human-AIDE interaction and interviewed 17 novices who programmed under time pressure with the GitHub Copilot AIDE. Using codebook thematic analysis, we synthesized themes from existing literature with interview data to produce a qualitative framework. Our results formalize the interaction patterns of **mindlessness** and **influencing**, describe the mixed quality of user interactions, introduce novices' anti-anthropomorphic **objectivizing** of AIDEs, and stress the importance of **prior knowledge** for success in AIDE interactions. We argue why CS educators should apply our research by teaching students to program effectively both with and without AIDEs.

3.1 Introduction

3.1.1 Background

Modern software development is highly language-based, providing several strong applications for Large Language Models (LLMs). A broad ecosystem of Artificial Intelligence-driven Development Environments (AIDEs) has arisen around this niche. AIDEs like GitHub Copilot, Tabnine, Amazon

Q Developer, and Continue are situated within traditional Integrated Development Environments (IDEs). Other AIDEs, like ChatGPT, Google Gemini, and Microsoft Copilot, are simply general-purpose chat-based assistants with capabilities useful for computer programming.

Increasing use and adoption of AIDEs is changing the role of humans in the software development process [14, 18, 36]. A great volume of the work that programmers previously undertook themselves can now be partially or completely outsourced [40] to AIDEs [3, 18, 23, 27, 43, 51, 52]. However, in computer science (CS) education, the true complexity of software development is artificially reduced into compact and learnable simplifications in course assignments [13, 41]. Consequently, AIDEs pose unique challenges in CS education [5, 24, 41], given they can handle the well-understood problem spaces typical of educational programming assignments with unusually high performance [16]. Varied opinions on whether educational value might outweigh potential risks have stagnated adoption of AIDEs in CS education [24, 41]. However, curricula that ignore AI may leave students ill-prepared for careers where they are expected to use AIDEs [24, 41].

3.1.2 Objectives and Contributions

To properly inform CS pedagogy, it is critical that research can better explain how novice programmers use and think about AIDEs [16, 40]. Many studies have provided qualitative analyses of AIDE use by programmers of varying ability levels [3, 18, 20, 21, 23, 27, 34, 40, 39, 43, 51, 52], but no work to date has performed a qualitative analysis based on this full body of existing literature. Finally, only a handful of prior studies [20, 23, 40] provide qualitative analysis focused specifically on novice programmers. Further, none of these enforced time pressure like that faced by students working on graded assignments [5].

To bridge these gaps, this work analyzes interview data according to the existing qualitative knowledge base of human-AIDE interaction. We collected this interview data after having 17 novices program with and without the GitHub Copilot AIDE under time pressure. Our investigation centered around the following research questions:

- **RQ1:** How do novice programmers and AIDEs interact while programming under time pressure?
- **RQ2:** What factors can explain how novices use AIDEs while programming under time pressure?
- **RQ3:** How does using an AIDE under time pressure affect a novice's thoughts and feelings?

3.1.3 Summary

Our qualitative framework ultimately included six topics relevant to human-AIDE interaction: **Interaction Patterns, Interaction Quality, Work Allocation, Appraisals, Anthropomorphism, and Prior Knowledge**. The selection of these six topics is described in Section 3.4. In addition to confirming themes derived from prior work, our analysis formalizes several new themes, revealing the objective, aimless, passive, and fraught interactions novices have with AI when programming under time pressure. We also demonstrate the positive impact of AIDEs and illuminate the road ahead for educators hoping to bring AIDEs to the classroom.

3.1.4 Statement of Theory

Despite value-drive and strategic, observant participation, novices programming under time pressure interact superficially, aimlessly, and passively with AI code generators, leading to subordination, suboptimal reliance, and underwhelming rapport.

3.2 Method

3.2.1 Literature Review Data Collection

We performed a targeted literature review to understand existing qualitative ideas relevant to our research questions. The inclusion criteria for a paper were as follows: 1) was a peer-reviewed publication, 2) used qualitative content analysis 3) pertained to using AI for programming, and 4)

reported data from human users (other than the authors themselves). While many researchers discuss and experiment with AIDEs, few recruit human participants as a part of their studies. Due to the even smaller number of studies exclusively recruiting novices, we chose not to make this a criterion for inclusion. The review began with two highly relevant papers: the seminal qualitative study of human-AIDE interaction by Barke et al. [3], and a more recent qualitative work studying novices by Prather et al. [40]. From there we used Google Scholar and Research Rabbit to perform snowballing [53] and periodic updating for work published during the analysis process. The first several papers were independently reviewed by the first author and two research assistants, who convened to discuss and refine the initial codebook. Further reviews were performed by the first author. In the end, the review consisted of 11 studies spanning three years and including data from 217 study participants and approximately 500 survey respondents or social media users. Because this review was an integral part of the data analysis process, we chose to defer introducing this work to Section 3.3 below. Additionally, reference to these studies is included as results in Section 3.4.

3.2.2 Experimental Interview Data Collection

To expand and evaluate our qualitative framework, we analyzed interview data related to our research questions collected as part of a previous study [17]. No data from the previous study was included in this analysis, and no data presented in this analysis has been reported previously.

Participants and Context

Seventeen undergraduate students taking Introduction to Programming (CS1) at a large US research university participated in the original study. The procedure entailed 80 minutes of programming in Python, divided between two trials with standard IntelliSense autocomplete and two trials with the GitHub Copilot Artificial Intelligence-driven Development Environment (AIDE). Participants were under time pressure to complete as many tasks as possible within each trial and were incentivized to perform their best through additional prize compensation awarded to top

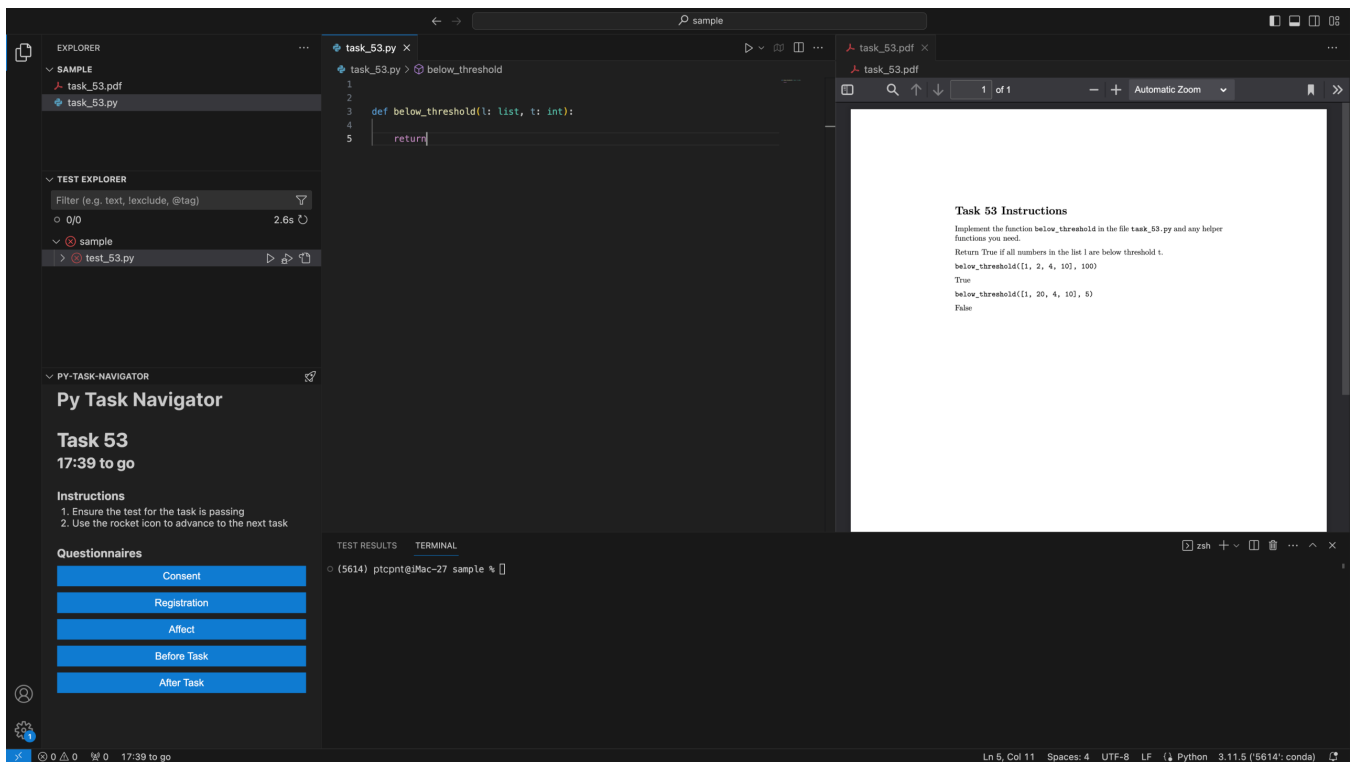


Figure 3.1: Interface seen by participants as they worked on Python programming tasks.

Note. Participants could see ghost text in their editor with inline suggestions from Copilot and could use a keyboard shortcut to open a popup tab with other suggestion options based on the text in their editor. The upper left panel or lower terminal allowed them to test their code. The lower left panel allowed them to submit their code and view the timer. The PDF file provided the task description.

scorers. They were trained to use the AIDE and shown both risks and benefits of using it, but were intentionally not given any guidance on how they should use it. Additional details on the study design can be found in the original publication [17].

Procedure

After completing all programming trials, participants were interviewed about their experience. The first author conducted all interviews using an interview guide written according to [4]. The questions that produced the data for this analysis are listed verbatim in Appendix A, but the interview style was semi-structured and not time limited. Member checking [28] was performed frequently throughout all interviews to facilitate valid interpretations during analysis. Interviews were audio recorded and transcribed first using automated transcription and then again by a research assistant.

3.2.3 Data Analysis

Preparation and Refinement

Each interview transcript was independently read and highlighted by the first author and at least one research assistant to select relevant and meaningful artifacts. The three analysts at this stage reviewed others' highlights and consulted to resolve disputes and generate a list of 466 artifacts. There was no balancing of the number of artifacts gleaned from each participant or size constraint on each artifact.

Codebook Thematic Analysis

We performed hierarchical [44] codebook thematic analysis (TA) [8], using an iterative process based on the deductive paradigm, but including an inductive step to go beyond existing work [9, 8, 15]. Codebook TA allows researchers to use themes as both inputs and outputs of the analytic process [8]. The goal of the analysis was to understand the data against a predetermined framework but acknowledge the subjectivity of the analysis process rather than striving for high

coding reliability. As such, the first author was the only analyst at the TA stage, an acceptable practice for codebook TA [8]. For further discussion on our efforts to mitigate threats to validity, see Section 3.6.

The process began with the initial codebook developed collaboratively during the literature review (see 3.2.1). All interview artifacts were then analyzed three times. Techniques employed included descriptive, axial, emotion, values, pattern, process, versus [47], and typological [44] coding. In addition to semantic (explicit) themes, we provide some extrapolation into latent themes insinuated by the tone or subtext of participants' words [9]. The first pass at the data was a familiarization step; the second was an initial coding and expansion of the codebook to cover novel ideas. Before the third pass, the literature review was revisited against new ideas, and codes were merged and organized. This was necessary because prior work often mentioned or observed ideas in passing that we formalized as part of our analysis, but not during the initial literature review. For the final pass, artifacts were re-labeled against the updated codebook, and artifacts describing or implying the manifestation (i.e. number of occurrences) of event-based themes were marked. Event-based themes (those under 3.4.1 Interaction Patterns, 3.4.2 Interaction Quality, and 3.4.3 Work Allocation) were themes whose significance differed based on whether they truly happened or were merely discussed. For example, if a participant were to say, "Copilot helped me code faster" (explicit) or "Copilot helps you code faster" (implied), we note this as an "occurrence" of **acceleration**. In contrast, the phrase "I can see how Copilot could teach you something new" would be an example of the *teaching* sub-theme, but not an occurrence. For non-event based themes (3.4.4 Appraisals, 3.4.5 Anthropomorphism, 3.4.6 Prior Knowledge), any matching artifact was considered equally important.

Summative Analysis and Reporting

We also performed summative analysis for the top level themes using frequency counting [35, 47] and employed matrix displays [33] to find the event-based themes that *occurred* most frequently (see 3.2.3 above). Topics that occurred and were discussed more frequently appear earlier in the Results. The organization of the codebook was based on thematic affinity, so high-level

themes were not necessarily more prevalent than lower-level sub-themes. For example, the **humanizing** theme consisted of just eight quotes, while the *evaluating by pattern matching* sub-sub-theme of **insourcing** alone consisted of 14. Readers are encouraged to consider the breadth of ideas catalogued in addition to the frequency of these ideas within our data. Well established codes were derived from prior work, and novel themes were named based on our interests in the implications of AIDEs on computer science education and the content of our unique dataset (not necessarily derived in vivo). Where a citation appears beside a code in the Results, that code was borrowed from the authors of the referenced work (e.g. in Table 3.1). Where possible, codes were formed as gerunds (i.e. process coding [47]), other nouns, or short phrases that most succinctly captured the themes’ meanings. References to thematic topics and top-level themes included in our framework are denoted by code words in **bold**, while lower-level sub-themes are shown in *italics*. For brevity, we use the word “theme” loosely in reference to the codes which denote the themes themselves. Strictly speaking, themes are not often fully explained by the code words [47], so readers should refer to the explanations for each code throughout the Results and referenced works.

Quotes in Section 3.4 were taken directly from transcribed interviews. Some quotes contain omissions (indicated by ellipses) or paraphrasing (indicated by *italics*), but never with the intention to alter the meaning of the original text. Quote selection statistics (see Appendix C) were tabulated and used during the writing process to balance representation of all participants to the best of our abilities.

3.3 Related Work

Our literature review focused on the qualitative portions of 11 peer-reviewed papers (see Table 3.1) published between 2021 and 2024. These papers mostly contributed qualitative-only analyses, but four [23, 27, 34, 51] had substantial quantitative analyses as well. Three studies [3, 40, 51] explored user interactions with the GitHub Copilot AIDE specifically, while one more [20] used GitHub Copilot along with several other off-the-shelf program synthesizers. Five other

studies [21, 23, 34, 43, 52] focused on evaluating or understanding user interactions with custom AIDE prototypes. Finally, two studies performed analyses of larger-scale survey or social media data [18, 27]. Aside from these two, all the studies recruited human participants for their data collection; the number of participants ranged from 11 [52] to 42 [43] (mean 24; median 28; total 217). Python was the predominant programming language, but the exact nature of programming tasks varied widely as did the qualitative techniques employed. A few high-level findings were common across several studies: 1) AIDEs are useful but also imperfect, 2) prompting AIDEs effectively is challenging, 3) users are tasked with evaluating AI-generated code, and 4) users over-rely on AIDEs despite holding moral stigmas against doing so. Because details from these studies are reported in the Results (Section 3.4), we focus here on the roles of time pressure and ability level in prior work.

3.3.1 Time Pressure

Only three studies enforced time pressure while users interacted with AIDEs. For two of these, participants were given synchronously timed programming tasks in 20 minute sessions [34, 51], while in [21], participants worked on two tasks asynchronously over one week. Other studies allowed participants to use AIDEs at their own pace. This design choice limits ecological validity, as both students and software professionals work under time constraints in the real world. Research in other domains has demonstrated how time pressure can cause users to rely more on AI, harming performance when the quality of the AI assistance is poor [42, 50]. One participant from [34] believed that the AIDE they used would have been more helpful if they had not been under time pressure. Jiang et al., who enforced asynchronous time pressure, suggested that a novelty effect may have caused users to engage more with the AIDE than they otherwise might have [21]. Having abundant time, perhaps they felt comfortable to fully explore and enjoy the AIDE's capabilities. By comparison, several participants in [51] "perceived a loss of control" using the AIDE, perhaps due to the synchronous time pressure they faced. Participants from all three studies enforcing time pressure reported some negative experiences trying to use or under-

stand AIDEs, suggesting that time pressure may indeed amplify negative experiences compared to conditions without time pressure [21, 34, 51]. Therefore, the current knowledge base is limited in that it focuses mostly on untimed, low-stakes interactions.

3.3.2 Ability Level

Within the novice level, three studies captured the range of programming experience: Kazemitabaar et al. recruited youth with no prior text-based programming experience [23], Prather et al. recruited CS1 students [40], and Jayagopal et al. recruited CS2 students. Other studies' data came from participants of unknown (in the case of [34]) or ranging ability levels, with a strong skew towards advanced programmers [3, 18, 21, 27, 43, 51, 52]. It is difficult to know which findings are general and which may be specific to users of certain ability levels, as the studies focused on novices [20, 23, 40] shared many findings with other studies. However, it does seem that novices relied heavily on AIDEs, struggled to interact with them effectively, and tended to use the simplest and most obvious features available [20, 23, 40]. An accurate qualitative model of novice interactions can draw on theory that applies also to advanced programmers, but should be validated against data specifically from novices.

3.4 Results

Our hierarchical codebook thematic analysis (see 3.2.3) yielded a multi-level qualitative framework co-constructed from both prior work and our interview data. Table 3.2 summarizes six identified topic areas and their associated top-level themes. For brevity, lower-level themes are omitted in Table 3.2 but further explored within each subsection. Figure 3.2 provides an overview of the framework in an idea map format, stressing the interconnected nature of the various high-level themes. Throughout the remainder of the Results, we discuss examples of each top-level theme from both prior work and our interview data, denoting topics and top-level themes in **bold** and sub-themes in *italics*.

Table 3.1: Summary of qualitative human-AIDE interaction studies.

Author	Date	n ^a	Tasks	TP ^b	Data Source	Analysis
Weisz et al. [52]	'21-04	11	fake code translation	no	interview	thematic analysis [9]
Vaithilingam et al. [51]	'22-04	24	Python programming	yes	questionnaire	informal
Jiang et al. [21]	'22-04	14	web programming	yes	interview, screen recording	inspired by grounded theory [19]
Jayagopal et al. [20]	'22-10	22	Python programming	no	think-aloud, interview	inductive thematic analysis, open coding
Ross et al. [43]	'23-03	42	Python programming	no	questionnaire, interaction logging	reflexive thematic analysis [46]
Barke et al. [3]	'23-04	20	Python, Rust, Haskell, and Java programming	no	interview, social media	grounded theory [49]
Scoccia [18]	'23-11	~100	none	n/a	social media	card sorting [54]
Prather et al. [40]	'23-11	19	C++ programming	no	think-aloud, interview	reflexive thematic analysis [8]
Liang et al. [27]	'24-02	410	none	n/a	survey	informal, open coding [47]
Kazemitabaar et al. [23]	'24-02	33	guided Python programming	no	interaction logging	deductive/inductive thematic analysis [6, 33]
Nam et al. [34]	'24-04	32	Python programming	yes	questionnaire	informal

^a number of participants, ^b Time Pressure

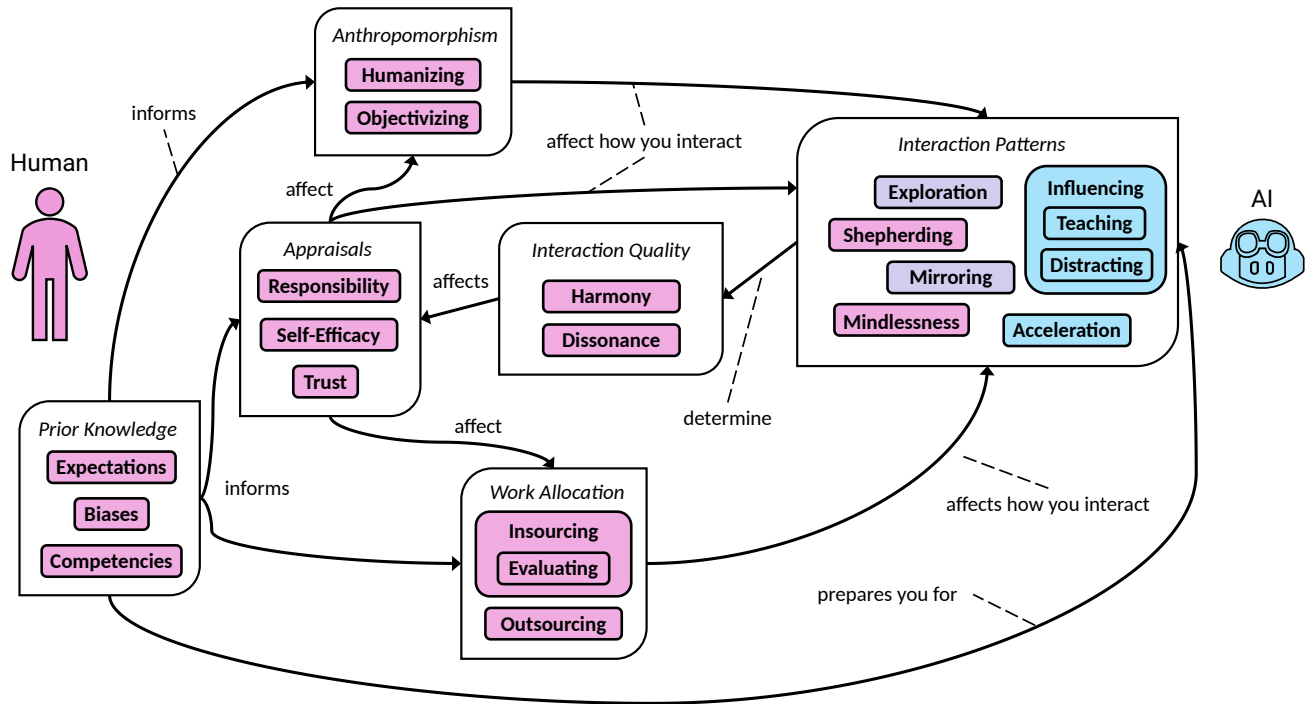


Figure 3.2: Idea map of qualitative framework resulting from our analysis.

Note. Human-centric themes are shown in pink. AI-driven interaction patterns are shown in blue, and purple interaction patterns are those initiated by humans but fulfilled by AI. A human enters an interaction with prior knowledge that informs their value appraisals, anthropomorphic understanding of the AI, work allocation strategies, and interaction patterns. Appraisals can also impact anthropomorphic understanding. Anthropomorphism, appraisals, and work allocation then affect the interaction patterns that will occur. Finally, the interaction itself determines the quality of the interaction for the human and helps them to update their appraisals.

The six *topic* areas were identified post hoc to better organize the disparate themes in our framework. The full hierarchy can be found in Appendix B: Qualitative Codebook for Chapter 3. The framework is presented here as a top-down outline of the hierarchical codebook according to prevalence within our interview data. Event-based themes that occurred often were given precedence over ones that were merely discussed often (see 3.2.3 for further detail). The topics we used to categorize the themes are as follows:

1. **Interaction Patterns**—describe how a human approaches and engages with an AIDE and how the AIDE affects the user in return. This topic is a frequent focus within prior studies (e.g. [3, 27, 40, 43, 52])
2. **Interaction Quality**—describes humans’ positive and negative experiences during or after using AIDEs. This topic was inspired by the ranging quality of users’ interactions with AIDEs across the literature (see Table 3.4), but also mirrors the positive and negative emotion sub-themes in [40].
3. **Work Allocation**—covers how a user divides work between themselves and the AIDE. Mastering this division is the key to unlocking the ever-improving power of AIDEs [36]. The origin of this topic is best credited to Prather et al., who formalized how users outsource work to an AIDE [40].
4. **Appraisals**—describe the systems of value humans use to assess their own abilities, to trust AIDEs, and to judge how to use AIDEs responsibly. We group these together because they are subjective value judgements rather than positive or negative experiences. While no prior work has grouped these ideas in this way, several studies have touched on each of these themes (see Table 3.6).
5. **Anthropomorphism**—explains how people think about AIDEs either as social agents or inanimate tools. This role dichotomy was introduced by Ross et al. in [43].
6. **Prior Knowledge**—pertains to how humans expect AIDEs to act, their competencies programming and using AIDEs, and what biases [31] they hold toward AIDEs. This topic helps

explain what gaps education needs to fill for novices to have successful AIDE interactions [41], and aligns well with guideline G5 of the 18 human-AI interaction guidelines in [1]: "ensure the experience is delivered in a way that users would expect, given their social and cultural context."

Table 3.2: Qualitative framework summary.

Topic	Theme	Prior Work	Description	Count	Unique
Interaction Patterns	acceleration [3]	[3, 20, 27, 34, 40, 43, 51, 52]	Programmers use AI to work more quickly and to maintain flow.	64	17
	shepherding [40]	[3, 18, 20, 21, 23, 27, 34, 40, 43, 51, 52]	Programmers apply deliberate techniques to coerce AI to help them reach their goals.	64	16
	influencing	[3, 27, 40, 43, 51, 52]	The availability of generative AI affects programmers' thoughts and actions while they work.	60	15
	mindlessness	[20, 27, 40, 43]	Novice programmers focus on actions and cues in the programming environment, without taking time to plan or think through problems deliberately.	35	15
	exploration [3]	[3, 18, 23, 27, 34, 40, 43, 51, 52]	Programmers use AI to find new ideas and approaches and to move forward when they reach an impasse.	42	14
	mirroring	[18, 21, 23, 27, 40]	AI generated content often reflects attributes of the context provided by the programmer, such as code correctness or level of sophistication.	5	4
Interaction Quality	dissonance	[3, 18, 20, 21, 23, 27, 34, 40, 43, 51, 52]	Interactions can cause negative reactions or feelings toward AI.	80	17

Topic	Theme	Prior Work	Description	Count	Unique
Work Allocation	harmony	[3, 18, 20, 21, 27, 34, 40, 43, 51, 52]	Users value and enjoy positive interactions with AI.	72	15
	insourcing	[3, 18, 20, 23, 27, 40, 43, 51, 52]	Preference and necessity drive users to do certain aspects of work themselves.	77	17
	outsourcing [40]	[3, 18, 21, 23, 27, 40, 43, 51, 52]	Users can offload work to AI.	80	16
Appraisals	responsibility	[18, 21, 40, 52]	Righteous use of AI demands restriction to certain conditions and considerations of various moral contingencies.	102	17
	self-efficacy	[34, 40]	Feelings about one's own abilities and worth stem from both the outcomes of programming efforts and the means used to achieve them.	29	13
	trust in AI	[18, 20, 40, 43, 51, 52]	Users depend on faith in AI's competence when they are unable to judge or predict its performance objectively.	8	5
Anthropomorphism	objectivizing	[18, 43, 52]	Through a breadth of experiences with AI, programmers develop a more neutral view of its ontological status.	52	16
	humanizing	[18, 40, 43, 52]	Users attribute human qualities to AI as a means to reason about its behavior.	8	7
Prior Knowledge	expectations	[3, 20, 21, 27, 40, 43, 52]	Lacking prior experience, users form predictions to anticipate AI interactions.	8	7
	biases	[21, 43]	Users are influenced by others' ideas about AI interactions.	5	4

Topic	Theme	Prior Work	Description	Count	Unique
	competencies	[20, 21, 40, 43]	Information a user has entering an AI interaction helps to guide their actions.	3	2

Note. Six topics help to organize the top-level themes from our hierarchical qualitative framework. Where a citation appears beside a theme, this code word is borrowed from the referenced work. "Prior Work" corroborating each given theme is also provided along with the "Description" of the theme used for analysis. The remaining columns provide the "Count" of excerpts from interviews labeled with a given theme as well as the number of "Unique" participant interviews with such excerpts.

3.4.1 Interaction Patterns

In this section, we discuss six **Interaction Patterns** that describe key aspects of how human-AIDE interactions take place. As shown in Figure 3.2, **shepherding** and **mindlessness** are human-centric patterns while **influencing** and **acceleration** are AI-driven patterns, and **exploration** and **mirroring** are initiated by humans but fulfilled by AI. Table 3.3 shows each of the six themes and their first-level sub-themes organized by the number participants who reported or implied an occurrence of the pattern (Event-Based → Count). For further information on the reporting of prevalence, refer to Section 3.2.3 and Table 3.3.

Acceleration

Barke et al. introduced **acceleration** as the first of two AIDE usage modes. In this mode, programmers work quickly and fluidly as the AIDE helps them carry out their own plans [3]. We expand this idea to include any instances where an AIDE is used for the purpose of efficiency or speed (see "speed" in [40]). For example:

Jayagopal et al. [20]: "these participants all initiated synthesis rapidly and were able to see synthesis results immediately"

Liang et al. [27]: ¹*Participants described "an AI programming assistant's ability to*

¹*Italics in extracts indicate paraphrased text, ellipses (...) indicate omitted text, and [brackets] indicate disambiguations.*

Table 3.3: Summary of prior work and data supporting interaction pattern themes.

Theme	Sub-Theme	Prior Work	Event-Based		Total	
			Count	Unique	Count	Unique
acceleration		[3, 20, 27, 34, 40, 43, 51, 52]	44	15	63	17
shepherding		[3, 18, 20, 21, 23, 27, 34, 40, 43, 51, 52]	46	14	64	16
shepherding	prompting	[3, 20, 21, 23, 27, 43]	21	9	25	9
shepherding	acquired skill	[3, 18, 20, 21, 34, 43, 51]	8	6	19	10
shepherding	adapting	[3, 18, 21, 23, 27, 40, 43, 51]	8	5	9	6
shepherding	backtracking	[3, 18, 23, 40, 51]	5	4	7	5
shepherding	drifting	[3, 18, 40, 51]	3	3	3	3
shepherding	slow accepting	[40]	1	1	2	2
shepherding	decomposition	[3, 20, 21, 23, 43]	0	0	0	0
mindlessness		[20, 27, 40, 43]	26	14	35	15
mindlessness	tunnel vision	[20, 43]	10	8	12	9
mindlessness	rushed	[34]	9	6	15	9
mindlessness	myopia	[40]	4	4	5	4
influencing		[3, 27, 40, 43, 51, 52]	23	11	60	15
influencing	distracting	[3, 40, 43]	14	9	24	11
influencing	variety	[3, 40, 43, 51, 52]	6	5	18	8
influencing	teaching	[27, 40, 43, 52]	2	2	21	11
influencing	style	[3]	1	1	2	2
exploration		[3, 18, 23, 27, 34, 40, 43, 51, 52]	29	10	43	14
mirroring		[18, 21, 23, 27, 40]	4	3	5	4

Note. Top-level themes, first-level sub-themes, corroborating prior work, and frequency counts are shown. When excerpts ("Count") or unique interviews ("Unique") indicated that the pattern occurred during the study, this was counted under "Event-Based". Any mentions of the patterns, inclusive of occurrences, were counted under "Total."

‘speed up...work’”

Our participants most commonly discussed interacting with Copilot for **acceleration**, especially while taking advantage of Copilot’s *inline completion* feature to autocomplete lines of code and comments.

Interview i16: “When I was doing [...] things with [...] a counter or something, it would just kind of instantly know how to do that, and that was helpful.”

Interview i19: “Even the ones I knew how to do myself, I can do it faster with Copilot which I liked.”

Interview i24: “There’s a test that asked me to [...] do something with prime numbers and [...] Copilot just did that [...] in a really short amount of time.”

Shepherding

Many of the strategies participants used to work more effectively with Copilot fall under the theme of **shepherding**. This theme was introduced in [40] but relaxed here to include similar frequently observed behaviors. By *prompting* [40] Copilot *with comments* and *with code*, participants provided context to improve suggestions as in prior studies.

Jiang et al. [21]: “participants used natural language by itself as their primary strategy” *and often reworded requests to get better results.*

Interview i11: “I would create a comment and at some points I would use part of the instructions, and I saw that Copilot was able to see what I was going for.”

Interview i12: “Giving it [...] information but not too much information- literally what you want it to give you, and less is honestly better.”

The data also strongly represent that **shepherding** is an *acquired skill*. When asked about reaching a rhythm using Copilot, several participants (i1, i12, i15, i16, i19, i43) said it took until

their second trial with Copilot for this to happen. They also shared speculations about how this skill might continue to improve with further use. These ideas align with findings of other prior studies discussing the importance of prompt engineering (e.g., [11, 18, 34]).

Nam et al. [34]: "Participants appreciated GILT's ability to easily incorporate their code as context, but some participants reported that writing a good prompt is still a challenge."

Scoccia, Hacker News Post 35853949 [18]: "Using ChatGPT to code effectively requires internalizing the small sliding context window and correct specification of the problem-space"

Interview i20: "I think if I used it more, I would get better at knowing when to use it"

Interview i42: "I definitely think I can get there with time." *It was also my first time using it; I just need to get used to it and how it works,* "just understand that a little bit better."

When Copilot's outputs were not correct, participants employed *adapting* [40] in attempts to move the code closer to being correct, as in other prior studies.

Kazemitabaar et al. [23]: "We identified 175 cases in which learners correctly modified the AI-generated code to align it better with the task's requested behavior."

Interview i4: *If the code* "generated an error [...] I'd be able to stop and correct it."

Interview i47: *You need to* "be able to [...] tweak it when it can't do what you want it to"

They also shepherded Copilot using *backtracking* [40], i.e. restarting a task or deleting portions of their work like in prior studies.

Vaithilingam et al., participant P1 [51]: "Because I did not understand several parts of the function [...] I did not know how to debug the function. This caused me to get rid of the whole function generated by Copilot and start over."

Interview i1: "If I get started on my own [...] and it ends up being not the correct way [...] the only way to get unstuck in that case would just be to restart and delete everything."

Interview i19: *If it didn't pass the tests* "I would try different strategies; maybe I'd try my own code and then let it fill in the rest [...] or I just tried a different solution."

Others mentioned behaviors consistent with *drifting* [40], which we categorize as a maladaptive form of **shepherding** rather than its own behavior altogether as in [40]. Therefore, "accept and adapt" *drifting* [40] is a type of *adapting* (see above), and ideas related to "deciphering" *drifting* [40] can be found between *overload* in 3.4.2 and *evaluating by inspection* in 3.4.3. The *drifting* pattern itself consists of users making repeated unsuccessful attempts to shepherd the AI, wasting time.

Scoccia, Hacker News Post 34703814 [18]: *Hallucinations* "can be particularly hard to detect, resulting in a frustrating experience: 'I burned an hour with ChatGPT'"

Interview i11: "I would go down a route with Copilot and it just didn't make sense [...] then I would go back, and I would try to do it another way, and sometimes it suggested the same pathway."

Interview i42: *I got sucked into paths that didn't work and...* "found it difficult to get out of that hole [...] usually [...] I'm pretty good at thinking of [...] a different way to solve something, but here [...] I couldn't think of anything."

Prather et al. described their participants using another form of **shepherding** they called *slow accepting*:

Prather et al. [40] "Here participants would type out Copilot's suggestion, often character for character, without outright accepting it (by pressing Tab)."

Participants i10 and i12 believed **shepherding** by *slow accepting* [40] to be a deeper way of engaging with suggestions than outright accepting. However, only i10 indicated that they used this strategy. Finally, we did not find examples of **shepherding** by *decomposition* [40] in any of our artifacts, despite abundant evidence of it in prior studies.

Ross et al., [43]: "Other participants approached the programming challenges in a piecemeal fashion, breaking them apart into a set of smaller tasks, then invoking the assistant to aid with each one."

Mindlessness

Participants often approached interactions with Copilot without careful planning or forethought, an idea we call **mindlessness**.

Liang et al. [27]: "Many participants reported not employing any strategy, as they found AI programming assistants to provide helpful suggestions without needing to perform specific actions."

Interview i9: *I probably wasn't using any strategies that I was "consciously thinking about [...] just things that [...] came in the flow."*

For example, three participants (i10, i11, i5) described not using certain features that might have improved their interaction with the AI simply because of *forgetting* about them. As in prior studies, many other participants experienced *tunnel vision* for the automatic inline suggestions from the AI, opting not to try more involved strategies like looking through the multiple completions panel.

Jayagopal et al. [20]: "Although the Copilot pane which shows additional suggestions (beyond the autocomplete one) displayed two correct synthesis outputs, the participant did not realize and continued with their strategy of renaming the function."

Interview i5: "I never actually went out of my way to look at the" *panel solutions on the right*; "I would just look at it if it was popping up as I was typing, and then see if that's what I wanted."

Interview i2: "I prefer to use [the inline suggestions]. I think it's kind of convenient to just tap Tab"

A similar challenge was *myopia*, a failure to think ahead or look at the bigger picture. Copilot can encourage short-sighted thinking, since it provides code that is likely to come next even when lacking complete context.

Prather et al. [40]: "Copilot mainly gives students feedback on 'Where to next?'"

Interview i15: "it gives me the answer and I already feel like I could have done that, but I'm taking the suggestion because I know [...] that's what happens next."

Interview i42: *Maybe my problem solving was slower with Copilot because* "when I do it by myself, I make more of a plan before and [with Copilot] I kind of just jumped right in."

Participants also reported *rushing* through the tasks due to time pressure, causing them to seek rapid forms of help from Copilot and engage more superficially with the code. Fittingly, this idea came up in one of the three prior studies which enforced time pressure.

Nam et al. [34]: "Participants prioritized completing more tasks rather than fully understanding the code, as reported by participant P18 [...] '[GILT] ... , which could definitely help one to tackle the task better if there weren't under the timed-settings.'"

Interview i16: "The timer [...] was stressing me out." *I was thinking*, "do I have enough time to look through these [panel suggestions]?"

Interview i43: "When Copilot solved it, it was nice because I was able to move on and not think about it."

Influencing

We identified **influencing** as a converse pattern to **shepherding**; AI can "steer" the user just as the user can "steer" [29, 52] the AI.

Weisz et al. [52]: "Meta-information [...] can help 'steer' human activity, complementing Louie et al.'s notion of how humans may "steer" the output of a generative model [29]"

Interview i12: "I feed Copilot and then Copilot feeds me, so it's just me going back on my own thing"

Oftentimes this influence came in the form of Copilot *distracting* participants from their own problem-solving approach, abandoning or changing it based on input from Copilot. There were three main types of distractions. The first was when Copilot was *polluting* a participant's thought process, corrupting their own plans.

Barke et al., participant P16 [3]: "I was about to write the code and I knew what I wanted to write. But now I'm sitting here, seeing if somehow Copilot came up with something better than the person who's been writing Haskell for five years."

Interview i10: "Sometimes it really confused me 'cause it was [...] telling me to do something and I would think [...] maybe I should do it this way."

Interview i47: "Your eyes are going to read it before you can think [...] you're just naturally going to gravitate towards the words that it comes up with. [It] distracts you from coming up with something that might be different."

The second type of distraction was *rerouting*, where Copilot changed a participant's plan but without necessarily compromising performance.

Interview i20: "With one where I used [Copilot] a lot [...] I was probably going to do it a different way, but I did adapt based on what it was giving"

Interview i15: "I tried taking its suggestions [...] just to see [...] where it would be going with that, but, well, its code just [...] didn't make sense to me"

The third type of distraction was Copilot *tempting* a participant to disengage with the task and outsource the work.

Prather et al., participant P13 [40] "it keeps filling up the screen with suggestions, and I'd be tempted to follow what it's saying instead of just thinking about it."

Interview i43: "I stopped thinking about [the tasks] as much and my first instinct would be to try to just get Copilot to do it, and then if it didn't work, then I would start thinking about the question"

Interview i9: "It's a lot more tempting to [...] not focus as much on learning [...] when you know that something else is already there."

Another form of **influencing** participants mentioned was exposure to a wide *variety* [43] of programming methods. This effect happens often during the **exploration** usage mode (see 3.4.1) but is a side effect of usage rather than a usage mode itself.

Weisz et al. [52]: "alternate translations had additional explanatory power in helping participants understand the nature of the underlying code"

Interview i4: "[Copilot is] pretty creative in the sense that it can think of things that maybe humans can't, because we have [...] one method or way of solving a particular problem, but [...] it can generate a lot of variations"

Interview i12: *A downside of giving Copilot too much information is that it would "branch almost, because [...] if you have a small place then it can only go [...] one stream, but if you have a big place to grow then it can go [...] four or five different places with it."*

The last major type of **influencing** discussed was *teaching*. Although only experienced during the study by two participants (i10 and i15), many others speculated about Copilot's ability to help them learn better. Even participant i13, who was generally unimpressed by Copilot, believed it could help them with learning.

Ross et al., participant P15 [43]: *It is hard to remember hundreds of APIs, but with AI I think "I'd learn new dev lang and new [libraries]/frameworks faster."*

Interview i10: "I know what max() is now. I think I would know by just [...] looking at it, but now I know it's [...] a command that I feel like I could use."

Interview i13: "I think [Copilot] would help with memorizing some functions" *and that would be more beneficial than harmful to learning because "you would still see [...] the function being typed out"*

Finally, two participants mentioned Copilot **influencing** their coding *style*, the form their code took—in one case (i12) the code was more complicated than it otherwise might have been, and in the other (i47), there were more comments due to *prompting*. Barke et al. mention six of their participants who demonstrated this idea [3].

Barke et al., participant P6 [3]: "The kind of comments I would write to Copilot are not the kind of comments I would use to document my code."

Exploration

Many participants used Copilot for **exploration** [3] of new ideas. The multiple completions panel was often helpful in this mode, though inline suggestions could be as well. Sometimes participants would accept the code they found, but other times, they were simply *consulting*.

Kazemitabaar et al. [23] "We identified 26 instances in which learners used the AI-generated code as an example [...] instead of replacing their code with the newly generated code."

Interview i19: *Usually I used Copilot when I had "written some code, didn't know how to proceed and then I would either used the inline feature, or find code from the panel"*

Interview i47: "There were [...] two instances where I went to the panel and compared a few." *I think it is pretty useful that the panel gives you "different kinds of solutions."*

Other than looking at new ideas, participants discussed exploration for the purposes of *getting started* on a task or *getting unstuck* from an impasse.

Vaithilingam et al., participant P1 [51]: "Copilot's function/line generation is a helpful reference; even if the generated code is not correct, it can point me in the right direction for completing the task."

Interview i1: "It was really good for giving me an idea as to how to get started"

Interview i2: *Maybe I would try only to use Copilot* "whenever I get really stuck on a programming assignment"

Mirroring

The last interaction pattern, **mirroring**, appeared in only a handful of artifacts. This is a phenomenon where the output of generative AI reflects the quality of the input. As far as we are aware, it was first discussed in the context of our literature review in [18] but is first named here.

Jiang et al., participant P4 [21]: "My first language is not English. I actually prefer to use the HTML language instead of typing the English because I'm worried about, 'What if I make a grammar mistake— will it still generate the code for me?'"

Interview i4: "There was one case where it didn't help because my code originally wasn't correct, and I changed my code from the beginning just a little bit, and then the answer that Copilot generated was good"

Interview i12: "if [...] I feed it wrong information or [...] I write something wrong, then it just makes it a lot harder"

3.4.2 Interaction Quality

In this section, we discuss the two sides of Interaction Quality programmers experience using AIDES: **dissonance** and **harmony**. Their sub-themes help to explain, from the human perspective, what causes an interaction to go well or poorly and how a user feels as a result. Table 3.4

shows first- and second-level sub-themes of dissonance and harmony organized by the number participants who reported or implied an occurrence of the given interaction quality (Event-Based → Count). For further information on the reporting of prevalence, refer to Section 3.2.3 and Table 3.4.

Dissonance

Participants very often experienced **dissonance** in interactions with Copilot. While we chose this code name, this theme has unsurprisingly been observed many times previously. The primary reason for **dissonance** was *misalignment* between participants' desires or needs and what Copilot would do.

Vaithilingam et al., participant P1 [51]: "Copilot generated a complete function to fulfill the full task, but part of the function did not work as desired"

Interview i5: "Normally when I get stuck, I'm just like really stuck and even having [...] Copilot wouldn't really save me (which it didn't in a lot of cases)."

Most of the time, this manifested as output that was *unhelpful* or caused *confusion*, such as when showing *unfamiliar* programming constructs.

Interview i13: "I thought it wasn't very useful for me for the tasks I was doing."

Interview i24: *Copilot gave concise code, but not in a good way*, "more like [...] harder to understand [...] what exactly it's trying to do"

Interview i11: "I would go down a route with Copilot and it just didn't make sense; there were parts of the language that I hadn't seen before and it confused me even more"

Some also complained that too much information from Copilot caused them to experience *overload*. This led them to disengage, because they did not want to spend time processing excessive content.

Table 3.4: Summary of prior work and data supporting interaction quality themes.

Theme	Sub-Theme	Sub-Theme 2	Prior Work	Event-Based		Total	
				Count	Unique	Count	Unique
dissonance			[3, 18, 20, 21, 23, 27, 34, 40, 43, 51, 52]	54	16	80	17
dissonance	misalignment		[3, 18, 20, 21, 23, 27, 34, 40, 43, 51, 52]	41	16	53	16
dissonance	misalignment	unhelpful	[18, 27]	13	9	15	9
dissonance	misalignment	confusion	[20, 27, 40, 51, 52]	13	8	15	9
dissonance	misalignment	overload	[3, 18, 20, 27, 34, 40, 43, 51, 52]	5	4	6	5
dissonance	misalignment	enabling	[40]	2	1	3	2
dissonance	misalignment	incorrect	[23, 27, 34, 43, 51]	1	1	3	2
dissonance	emotional		[21, 40, 51]	8	5	10	6
dissonance	mirage		[3, 18, 43, 51]	2	2	4	4
dissonance	apprehension		[27, 40]	1	1	13	7
harmony			[3, 18, 20, 21, 27, 34, 40, 43, 51, 52]	49	14	72	15
harmony	alignment		[3, 18, 20, 40, 43, 51, 52]	27	12	42	13
harmony	alignment	helpful	[3, 18, 40, 43, 51, 52]	12	10	21	12
harmony	alignment	clarity	none	5	5	7	7
harmony	alignment	easier	[20]	9	6	12	6
harmony	emotional		[3, 20, 21, 40]	7	6	8	6

Note. Top-level themes, first-level sub-themes, corroborating prior work, and frequency counts are shown. When excerpts ("Count") or unique interviews ("Unique") indicated that the pattern occurred during the study, this was counted under "Event-Based". Any mentions of the patterns, inclusive of occurrences, were counted under "Total."

Barke et al. [3]: *Participants* "P4, P6, P7 and P13 did not like the multi-suggestion pane popping up in a separate window stating that it added to their cognitive load."

Interview i24: *Some of Copilot's suggestions were* "super long and complex, and I didn't really take the time to [...] try and analyze like, 'oh, what is it trying to do?'"

Interview i43: *I didn't like the panel as much as the inline, because for the panel suggestions* "I would have to [...] read through and see what each one would do"

Errors and bugs in code were another cause for **dissonance**. Not only did participants mention issues with Copilot's code being *incorrect*, but some were also bothered by its *enabling* issues in their own code.

Ross et al., participant P28 [43]: "It makes mistakes often enough to be not very practical."

Interview i11: "I can tell that Copilot does have some errors, so you definitely have to double check."

Interview i5: "[Copilot is] not super helpful in terms of, like, it won't tell you if you're totally going on the wrong track. Or if there's [...] a way better way to do things"

In addition to *misalignment*, a few participants also mentioned how **dissonance** could be *emotional* reactions like shame or emptiness.

Interview i15: *Talking to Copilot through the comments* "blew my mind; I was like, [...] 'this feels like a violation of the honor code.'"

Interview i11: "I was more negative and upset when I used Copilot. [...] When I finished the code by myself, there was this achievement that I felt, this goal that I met, but with Copilot I didn't. It wasn't there. It was empty."

Another form of *emotional dissonance* was frustration, a common feeling discussed by participants in prior studies.

Jiang et al., participant P5 [21]: *Using the AI is a "combination of fun and frustrating."*

Interview i42: "I feel like I was more frustrated" *when I worked with Copilot*, "because I was [...] stuck and [...] I hate that feeling [...] I had [...] this robot helping me, and I couldn't continue even with the robot"

Four participants (i1, i10, i43, and i47) discussed the idea of **dissonance** due to a *mirage*, where AI-generated code deceptively appears correct. However, only two specifically mentioned experiencing it despite its appearance in prior work.

Scoccia, Hacker News Post 35765407 [18]: "nearly every time I've asked ChatGPT for code, it gives me either plausible-but-wrong code, or total hallucinations"

Interview i1: "Sometimes [Copilot] would give me a solution that looks almost correct [...] and I would test it, and it wouldn't work [...] that kind of makes it harder when it does do that."

Interview i10: *It was confusing when I would change my plan based on Copilot's suggestions*, "but then it wasn't working."

Finally, while participants were not generally fearful about Copilot, some discussed how *aprehension* might be reasonable based on their interactions, like in prior studies.

Prather et al, participant #15 [40]: "It might take over the world and write its own code."

Interview i20: "I think it might be [...] slightly unsettling [...] how well it can predict things, but I don't think it's [...] a real fear or concern."

Interview i43: *Sci-Fi does make me* "a little concerned [...] I wouldn't say I'm seriously concerned about it, but it's [...] something to think about." *In movies you always see that classic scenario of "robots taking over and [...] controlling life."*

Harmony

Despite their negative experiences, participants also mentioned many examples of **harmony** in interactions. Like **dissonance**, we chose the code name for this theme, but it has been observed many times previously (e.g. [3, 18, 21, 40, 51]). As in prior work, the most common type of **harmony** was related to *alignment* between participant's desires and Copilot's behavior such as when participants found suggestions *helpful*.

Scoccia, Hacker News Post 36091502 [18]: "I find chat GPT to be very helpful for working with programming languages that I'm less comfortable using"

Interview i2: "I think Copilot helped me a lot in solving the problems."

Alignment also happened when Copilot provided good *clarity* or made programming *easier*.

Interview i19: *Copilot's suggestions were fairly easy to understand.* "I like the fact that they included comments themselves, so it makes it easier to read."

Interview i24: *Using Copilot* "made things a lot easier."

Harmony was sometimes exhibited by mention of *emotional* experiences like calm, enjoyment, and fun. However, participants' positive emotions were generally more tied to their success on the programming tasks than on whether they used Copilot (see *teleology* in 3.4.5).

Jayagopal et al., participant P19 [20]: "I really liked how I could see the suggestion without having to do anything."

Interview i19: *Compared to working alone, with Copilot, there's* "less mental taxation on you so you're more calm, but [there's] not a super big difference."

Interview i9: *For my classes,* "It'd be nice to have Copilot. I think I would really enjoy using it."

3.4.3 Work Allocation

In this section, we describe the two behaviors of **Work Allocation** undertaken by a programmer using an AIDE: **insourcing** and **outsourcing**. We identify to whom certain work is generally allocated and why one allocation behavior might be chosen over the other. Table 3.5 shows first- and second-level sub-themes of each of the two behaviors organized by the number participants who reported or implied an occurrence of a given behavior or reasoning (Event-Based → Count). For further information on the reporting of prevalence, refer to Section 3.2.3 and Table 3.5.

Insourcing

When a user does work themselves, rather than having or letting AI do it, we call this **insourcing**. This behavior is first named here (inspired by [40]) but was observed many times previously. The work participants talked most about **insourcing** was *evaluating* AI generated code. The novices in our study relied heavily on *pattern matching* [3] to decide whether the code they were *evaluating* matched what they expected they might need.

Barke et al., participant P6 [3]: "When I see a suggestion, the closer that suggestion is to the mental image I hold in my head, the more likely I am to trust it"

Interview i10: *I would decide to take one of Copilot's suggestions* "if it's something that I would [...] write on my own"

Interview i5: *Often*, "I had a very specific way I wanted my code to work, and then I didn't really know what [Copilot] was doing"

However, some also used more careful *inspection* to decide whether the code looked correct.

Jayagopal et al. [20]: "Novices spend time reading and tracing synthesized programs rather than accepting them without question."

Interview i1: *Sometimes after I took an inline suggestion*, "I would look at it for a bit."

Table 3.5: Summary of prior work and data supporting work allocation themes.

Theme	Sub-Theme	Sub-Theme 2	Prior Work	Event-Based		Total	
				Count	Unique	Count	Unique
insourcing			[3, 18, 20, 23, 27, 40, 43, 51, 52]	42	15	77	17
insourcing	evaluating		[3, 18, 20, 23, 27, 40, 43, 52]	23	12	34	15
insourcing	evaluating	pattern matching [3]	[3, 27]	12	8	14	9
insourcing	evaluating	inspection [27]	[3, 20, 40, 52]	5	5	8	6
insourcing	evaluating	executing [3]	[3, 27, 52]	7	6	8	6
insourcing	evaluating	tinkering [23]	[23, 43]	2	2	6	5
insourcing	problem solving		none	4	4	5	5
insourcing	under-reliance		none	5	2	8	3
insourcing	control		[27, 51]	2	2	3	3
insourcing	slow rejecting		none	3	2	3	2
insourcing	goal setting		none	1	1	6	6
insourcing	understanding		none	1	1	9	5
insourcing	creativity		none	0	0	3	3
insourcing	fun		none	0	0	1	1
outsourcing			[3, 18, 21, 23, 27, 40, 43, 51, 52]	37	13	80	16
outsourcing	expression		[3, 23, 27, 40, 43, 52]	8	6	14	8
outsourcing	expression	syntax	[23, 27, 40, 43]	3	2	5	3
outsourcing	expression	typing	[3]	3	2	5	3
outsourcing	over-reliance		[3, 23, 40, 43, 51, 52]	15	8	42	14
outsourcing	over-reliance	not understanding	[40, 51, 52]	12	6	14	7
outsourcing	over-reliance	not learning	[3, 40, 43]	1	1	10	8
outsourcing	simple work		[3, 21, 23, 27, 43, 51, 52]	4	4	13	8

Note. Top-level themes, first-level sub-themes, corroborating prior work, and frequency counts are shown. When excerpts ("Count") or unique interviews ("Unique") indicated that the pattern occurred during the study, this was counted under "Event-Based". Any mentions of the patterns, inclusive of occurrences, were counted under "Total."

Interview i2: *I would look through "all the possible solutions from Copilot." Then I would take one of those and "just try to understand it by reading through it by myself."*

Because the novices in our study did not always clearly distinguish between running [23] code and testing [23] it, we lump these together as *executing* as in [3].

Liang et al. [27]: *69% of participants reported evaluating AI suggestions by "executing the code"*

Interview i42: *I found it "helpful [...] seeing how the test cases went wrong."*

Interview i15: *When I get a big chunk of text, I can't tell what is happening; "I just want to [...] take it and see if it works."*

A few of our artifacts also fell under the theme of *tinkering* [23], an idea we expand beyond using temporary changes for better understanding code [23] to encompass the debugging process as a whole. We suspect that participants engaged in much more of these behaviors than they talked about in their interviews. Also, when participants specifically mentioned changing code during the debugging process, we categorized that instead as **shepherding** by *adapting* (see 3.4.1).

Kazemitabaar et al. [23]: *"We identified 30 instances in which learners actively tinkered with the AI-generated code by temporarily changing values, conditions of while loops, modifying syntax, or even temporarily removing code to see how it contributed to the entire program."*

Interview i19: *When I was working on the tasks, "I would look at the terminal to find the error. I did that often."*

Interview i16: *I think that in the future, "Copilot could write the code, but then a person would have to go in and [...] find all the exceptions and stuff and debug it."*

Other types of work participants discussed **insourcing** were *problem solving*, *goal setting*, *understanding*, and *creativity*.

Interview i4: "I do prefer coding alone just because it allows me to think more [...] Copilot sort of takes away the thinking process a little bit"

Interview i9: *When I decided not to take a panel suggestion, it was because* "it didn't necessarily suit what I was trying to code for."

Interview i10: *I wanted to understand Copilot's code because* "I feel like I wouldn't be able to [...] add on to it if I didn't understand it."

Interview i47: *A lot of times AI generated content can be* "lacking in tone, [...] very lackluster, very [...] generic and [...] you have to, like, be able to, like provide some sort of like expertise to it."

Two participants (i9 and i43) used an **insourcing** behavior we call *slow rejecting* (inspired by [40]), where they saw a suggestion from Copilot but typed something else to implicitly reject it. Two others (i10 and i20) exhibited *under-reliance*, describing how they failed to get the full benefit out of Copilot due to **insourcing** too much work. When reflecting on why to insource work, participants cited *fun* and *control* as motivators.

Interview i10: I think I would want to try to code on my own before using Copilot. "I feel like it's more fun that way"

Interview i42: "I had my way and I wanted to stay with that."

Outsourcing

Outsourcing, which was introduced by [40], occurred when participants allowed or requested AI to do work for them. The most common type of work participants discussed **outsourcing** was *expression* of ideas into code. In prior work, participants mentioned using AI for *expression* of template or baseline code that they could then change.

Weisz et al. [52]: "many participants expressed a willingness to accept imperfectly-translated code as a starting point"

Though sometimes *expression* helped with logic, it also helped with writing correct *syntax* and reduced the need for *typing*.

Ross et al., participant P40 [43]: "It helps me to avoid silly syntax errors and can [help] when I cannot remember exact function/method names and required arguments"

Interview i16: "You don't have to think of the code as much on your own. Especially things like syntax [...] you don't have to be as particular"

Interview i47: "I tried to [...] reserve it for [...] cases where [...] I know exactly what I need to do. I just can't be bothered to [...] type it all out."

As in prior work, we identified many artifacts consistent with *over-reliance* on AI assistance.

Vaithilingam et al. [51]: "we observed eight instances of over-reliance on Copilot. For example, [participant] P8 simply accepted the generated code and said, 'I guess I will take its word.'"

Interview i5: "There was [...] one where I just really didn't even have to look at the question; [Copilot] just totally knew what to do."

Our participants recounted their *over-reliance* on Copilot during the programing trials, such as by using code but *not understanding* how it worked or *not learning* the constructs within it.

Interview i24: "With some of the tasks that I was doing, I wasn't really [...] trying to understand some of them; I was [...] just entering in what Copilot was suggesting."

Interview i15: *Even though I saw Copilot use it*, "If I needed to use the insert function, I'm not quite sure how to do it [...] I probably wouldn't think of it."

Lastly, participants talked about **outsourcing** of *simple work* like repetitive tasks and basic solutions.

Weisz et al., participant P4 [52]: "Focus on letting the machine do the stuff that's really easy"

Interview i9: *I used the inline suggestions* "a decent amount, especially for [...] the shorter functions or the ones that didn't require that much [...] text."

Interview i43: *In a real-world situation*, "It'd be much more important to actually think about how you want to do it, and then let Copilot autofill more of the [...] repetition type things"

3.4.4 Appraisals

In this section, we describe three types of value **Appraisals** we encountered between prior work and our data. *Responsibility* appraisals are those programmers use to inform what types of interaction choices are morally upstanding, while **self-efficacy** and **trust in AI** are appraisals (by the human) of the interaction participants themselves. Table 3.6 shows first-level appraisal sub-themes organized by the number of participants who reported the given appraisal (Count). For further information on the reporting of prevalence, refer to Section 3.2.3 and Table 3.6.

Responsibility

Every participant had something to share relating to the theme of **responsibility**, the idea that there is a right or necessary way to approach use of AI code generators. There were many artifacts stressing the importance of *participation* by the human user. In contrast to the dividing of work discussed in 3.4.3 above, *participation* refers to the value of the human being involved in a collaboration with AI.

Weisz et al., participant P6 [52]: "I might want to have the option to look into what the AI is doing just so I can interpret or prepare my code better for the translation."

Table 3.6: Summary of prior work and data supporting appraisal themes.

Theme	Sub-Theme	Prior Work	Count	Unique
responsibility		[18, 21, 40, 52]	102	17
responsibility	participation	[18, 40, 52]	60	17
responsibility	context	[18]	17	9
responsibility	integrity	[40]	10	6
responsibility	mindset	none	8	6
responsibility	accountability	none	8	5
responsibility	maturity	[21]	5	2
responsibility	moderation	none	2	2
self-efficacy		[34, 40]	29	13
self-efficacy	low	[34, 40]	16	10
self-efficacy	high	none	8	6
trust in AI		[18, 20, 40, 43, 51, 52]	8	5

Note. Top-level themes, first-level sub-themes, corroborating prior work, and frequency counts are shown. Only the number of interview excerpts ("Count") number of unique interviews ("Unique") are reported, as appraisals reflect value systems rather than events.

Interview i24: *I might be concerned if this AI were a lot better, but right now I'm not, "because it needs [...] your input in order to work."*

Interview i47: *You just had to "explain 80% of what it needs to do, just in the comments alone. If for some reason [...] those aren't clear, then you had to [...] start coding it yourself."*

Many participants also felt that the *context* in which an AI code generator is used has a bearing on the appropriateness of doing so.

Scoccia [18]: *Some commenters feel "that ChatGPT (and other large language models) will soon be able to replace software developers, especially in domains that are not safety-critical: 'I think it could go either way depending on the product' " (Hacker News Post 35205999).*

Interview i13: *"A company versus school is very different [...] when you're in school,*

you're trying to learn, but in a company [...] you kind of just want to do that thing, and so if ChatGPT can help a little, I think it'd be fine."

Interview i43: "I think if [Copilot] was allowed, I would use it for code I already understand, but if I came across a problem that I didn't, I would try to figure it out myself."

Several felt that using AI code generators might threaten their *integrity* or academic honesty.

Prather et al. [40]: "Students [...] were uncertain about the implications [of Copilot] for plagiarism."

Interview i47: *If I can use Copilot to "get the boring parts out of the way, that's fine," as long as "no one's [...] relying on me to do it by myself."*

Interview i15: "I don't want to use it [...] to my full advantage [...] I don't want to get an honor code violation."

Others described how the *mindset* a programmer takes while programming and the *maturity* of the programmer can determine how good or responsible the usage is.

Interview i24: *I might only analyze Copilot's suggestions more deeply "if I had the motivation to learn what it was trying to do, or [...] how to do the things that it was suggesting"*

Jiang et al., participant P12 [21]: "I can see it being of great use to people who are not that proficient in coding to be ... a good sort of introduction, or to really ... lower the barrier to entry for coding"

Interview i42: *Right now, I'm still learning, so "I want to come up with my solutions myself [...] but one day once [...] these really simple codes become [...] automatic [...] I might as well have a program like this writing them for me."*

In addition to integrity, several participants cited *accountability* as a critical value to uphold while using AI to code. That is, they believed a programmer should be able to answer for the code that they deliver. Two participants (i11 and i47) thought it was okay to use AI to code in *moderation*.

Interview i5: "I want to be the authority on my own work [...] I were to ever have someone [...] ask me about it or I need to modify something, I'd want to be able to like fully understand what I had submitted."

Interview i11: "I think that trying to [code] by myself as long as I can would be better, but if I need a suggestion, one little suggestion (not the whole project) that could be helpful."

Self-Efficacy

Self-efficacy is a reflexive appraisal of one's own capabilities and potential to succeed at something. It was first discussed in our context in [40], but we provide concrete examples of it in action. Over half of the participants exhibited some hint of *low self-efficacy* in relation to using Copilot. Often, this was because participants did not feel they could take credit for the success they achieved with Copilot's help.

Interview i11: *When I used Copilot*, "I felt like I gave up more quickly [...] I relied on copilot too much, which I was not happy about." *Copilot would make me more efficient, but I think my confidence in CS* "dropped a bit, 'cause isn't the point [...] being able to do it on your own?"

Interview i47: "I don't think it could ever be satisfying to use Copilot [...] I don't know how something can be gratifying when [...] something does something for you."

Not only did participants struggle to take credit for success with Copilot, but subtle comments indicated their *self-blaming* when they encountered struggles with Copilot.

Nam et al., participant P28 [34]: "I think I just didn't word the question well."

Interview i2: *I don't think Copilot ever made it harder to program*; "I just have to [...] explain the question more clearly, and then [...] debug [...] what Copilot gives me."

Interview i9: *When* "Copilot wasn't necessarily giving me what I was looking for [...] there was a little bit of friction, but I wouldn't consider it friction between me and Copilot; I just consider it [...] friction between, like, my thinking."

A few other comments about self-efficacy did not quite align with *low* or *high self-efficacy*. In these cases, participants describe that programming alone is more satisfying than using Copilot but not that using Copilot is unsatisfying. However, they do shed light on how *over-reliance* (see 3.4.3) might degrade **self-efficacy** over time.

Interview i19: "It was less satisfying on the [tasks] that I didn't do myself," *but there were also ones* "I did myself, and I thought through, and then adding Copilot" *to speed things up* "doesn't make it any less satisfying."

Interview i43: "It's more satisfying to come up with a solution yourself [...] just a sense of accomplishment."

On the other hand, several participants' comments indicated *high self-efficacy* in relation to using Copilot. Some felt that they deserved credit for success with Copilot when their **shepherding** played an integral role. Others felt that assistance and stimulation from Copilot would improve their confidence throughout their CS learning path.

Interview i12: *It was a good feeling when Copilot helped me get unstuck, but really* "it was just me [...] giving it [...] the correct prompt"

Interview i20: *Knowing I could get more practice by using Copilot would help my confidence in CS, because if* "I were coding randomly in a file [...] I don't know where I could go with that [...] but with Copilot I guess it could give me different directions or things I could try [...] it's kind of like a stimulus in that way."

Trust in AI

Only five participants shared comments that related to trust or confidence in Copilot (i2, i24, i43, i4). Both prior work and the limited data we have suggest that users place some level of initial **trust in AI**.

Prather et al. [40]: "Novices tend to view feedback in programming contexts to be the truth and the systems generating it to be infallible [26]"

Interview i2: "I believe [Copilot] has like a better programming sense compared to me."

Interview i43: *I didn't usually try to fully understand Copilot's suggestions; "I just kind of trusted Copilot." I think of it like a "database of lots of good code," so I would assume that its solutions are "probably better solutions than I would come up with for the most part."*

Then this initial level is either reinforced through **harmony** or broken by **dissonance** over time.

Vaithilingam et al., participant P7 [51]: "At this time I probably prefer IntelliSense just because I trust my own Googling and understanding code examples online rather than opaque suggestions from Copilot"

Interview i24: *After working with Copilot, "I'd say I feel [...] better about Copilot and it's suggestions than I did before."*

Interview i4: At some point "I lost a bit of confidence in Copilot because I was trying to try out different things, but [...] Copilot was generating the same answer."

Interview i20: Copilot can "be distracting just in deciding which approach to take," but I think I can get better at knowing "when to trust what I am going with"

3.4.5 Anthropomorphism

In this section, we describe the opposing perspectives programmers can take with respect to AIDES' **Anthropomorphism**, i.e. whether it is an inanimate technical object (**objectivizing**) or a human-like character (**humanizing**). In addition to the plain content of participants' comments, subtext, especially, helped to capture these perspectives. Table 3.7 shows first-level sub-themes organized by the number of participants whose comments aligned with the given theme (Count). For further information on the reporting of prevalence, refer to Section 3.2.3 and Table 3.7.

Table 3.7: Summary of prior work and data supporting anthropomorphism themes.

Theme	Sub-Theme	Prior Work	Count	Unique
responsibility		[18, 21, 40, 52]	102	17
responsibility	participation	[18, 40, 52]	60	17
responsibility	context	[18]	17	9
responsibility	integrity	[40]	10	6
responsibility	mindset	none	8	6
responsibility	accountability	none	8	5
responsibility	maturity	[21]	5	2
responsibility	moderation	none	2	2
self-efficacy		[34, 40]	29	13
self-efficacy	low	[34, 40]	16	10
self-efficacy	high	none	8	6
trust in AI		[18, 20, 40, 43, 51, 52]	8	5

Note. Top-level themes, first-level sub-themes, corroborating prior work, and frequency counts are shown. Only the number of interview excerpts ("Count") number of unique interviews ("Unique") are reported, as anthropomorphisms reflect perceptions rather than events.

Objectivizing

We identified that participants generally created **objectivizing** conceptualizations of Copilot, classifying it as a neutral technical object rather than a persona.

Interview i20: "Overall my feelings are probably neutral; I think it's helpful for users to get practice with it and learn how to use it well to their advantage."

Many objectivizing comments simply displayed participants' emotional *dispassion* toward Copilot or AI in general.

Interview i13: *When Copilot can't help* "I wouldn't say it's [...] super disappointing," *and when it did* "I was just like, 'Okay.' [...] It was enjoyable [...] but [...] it wasn't [...] mind blowing or anything."

Interview i5: It didn't really get in my way, because "I'm not that good of a coder, so it's not like I'm like, 'Oh my gosh, it's messing my rhythm up' "

Many comments also captured the *inconsistency* of experiences with Copilot; sometimes it was very helpful, while other times it was not.

Ross et al., participant P19 [43]: "The code generator was good but you still have to really check it."

Interview i12: "I think [working with Copilot] was good in some cases, but not so good in other cases, especially" *on the* "tasks where [...] I didn't fully understand the question."

Interview i16: "It was either like Copilot just solved the problem for me and I didn't do anything or I just kind of solved it and then [...] along the way [...] Copilot would help me"

Several participants referred to Copilot as a tool or *resource* for them to use rather than personifying it.

Interview i9: *It was comforting to know that Copilot was* "kind of like another resource that I could use"

Interview i4: "I think it was worse [programming] alone [than with Copilot], because that's one less tool that I was able to utilize"

A handful of **objectivizing** comments demonstrated subscription to a *teleology* of programming also present in prior work—that the means used to write code only matters insofar as it progresses the user towards successfully achieving an end goal.

Weisz et al., participant P2 [52]: *I would "be happy with [the NMT model]" as long as "I get the correct output."*

Interview i19: "I think my feelings stayed pretty consistent throughout—a little bit better when you have more success with it."

Interview i47: *The only triumph I had was "getting to [...] pick myself back up after" getting stuck on one. I got a similar question later and "I was like, 'Okay, I'm not an idiot [...] I do know this.'"*

In line with prior studies, two participants (i1 and i12) objectivized Copilot by *not blaming* it when things went poorly in interactions with it.

Scoccia, Hacker News Post 36087083 [18]: *What ChatGPT gives you is "not perfect code, but it's the right idea."*

Interview i1: *When Copilot leads me astray and I have to restart, "it depends on probably what the problem is and how familiar I am with it, but I probably would try to use it again."*

Interview i12: *Sometimes Copilot didn't help, but "if I don't understand the question then, like, I don't know what I want, so Copilot can't know what I want, right?"*

Humanizing

There were several cases of participants **humanizing** Copilot by implying it to have human-like traits, as in prior work. This theme was a relatively rare way to think about Copilot with just under half of our participants doing so.

Prather et al. [40]: "several participants reported that they felt that Copilot was aware, knowledgeable, and had agency."

Interview i15: "It felt like a little sibling, you know, they're like, 'let *me* help.' [...] I want to consider it my friend, but I'm a little scared, you know."

Interview i2: "I guess I can just explain everything to it, so that it can understand me more clearly."

3.4.6 Prior Knowledge

In this section, we discuss the final topic: **Prior Knowledge**. As this topic is considerably smaller than others, its top-level themes are more comparable in size to other topics' first-level sub-themes. However, we emphasize this portion of the results for its centrality to the engineering of human-AIDE interaction. Beyond improving AIDEs themselves, education is a primary mechanism for molding human-AIDE interactions to match individual, organizational, and societal standards. Table 3.8 shows first-level sub-themes organized by the number of participants whose comments aligned with the given theme (Count). For further information on the reporting of prevalence, refer to Section 3.2.3 and Table 3.8.

Table 3.8: Summary of prior work and data supporting prior knowledge themes.

Theme	Sub-Theme	Prior Work	Count	Unique
expectations		[3, 20, 21, 27, 40, 43, 52]	8	7
expectations	underestimating	[20, 43]	4	3
expectations	overestimating	[3, 41, 52]	1	1
biases		[21, 43]	5	4
biases	training	[21]	2	2
competencies		[20, 21, 40, 43]	3	2

Note. Top-level themes, first-level sub-themes, corroborating prior work, and frequency counts are shown. Only the number of interview excerpts ("Count") number of unique interviews ("Unique") are reported, as these themes refer to participants' past experiences rather than interaction experiences.

Expectations

Participants were asked directly about how their *expectations* compared to what they experienced. Their answers reveal that their preconceptions guide their actions and sometimes fail to match reality.

Liang et al. [27]: "Participants developed mental models of these tools [10], as they reported leveraging existing code as additional data for the AI programming assistant to use"

Interview i11: *Between programming* "on my own versus doing it with Copilot, I realized I liked it more by myself [...] which I was not expecting, but happy to know that now."

Interview i13: *Copilot* "didn't help that much, but I thought it would help, so that was an interesting contrast."

Three participants (i19, i20, i47) described *underestimating* Copilot's abilities. As in prior work, this can happen when users have not yet experienced the benefits of a new tool.

Jayagopal et al., participant P6 after using Flash Fill in Excel [20]: "Oh, cool! That was easy. I didn't know for sure if it could figure out the pattern from [one example]."

Interview i47: *Things* "went pretty much as expected [...] maybe slightly better [...] it needed less than I thought it would [...] going off the comments alone was kind of insane."

One participant (i1) described *overestimating* Copilot's abilities, which happens when users are hopeful about an AIDE's utility.

Barke et al. [3]: "[Twitch streamer] S3 expected the multiple suggestions to be diverse and was sometimes disappointed when they were not."

Interview i1: *I thought* "if I were to just type a few [...] basic instructions, it'd be able to do the rest for me, which it was not."

Biases

Participants also described *biases* they had towards AI, such as i43's encounters with Sci-Fi movies (see *apprehension* in 3.4.2) or i15's having seen a video on social media critiquing generative AI in the popular productivity app, Notion. Similarly, participants in prior studies have been biased entering into interactions with AIDEs.

Ross et al. [43]: "People told us about their frustrating experiences with conventional chatbots and raised doubts about the knowledge, capabilities, and value of a conversational programming assistant."

Two participants (i12 and i13) mentioned being biased by incorrect AI behavior shown in the **training** videos for the study. Similarly, Jiang et al. mention how *training* can help to bias users' *expectations*, too.

Jiang et al. [21]: "in onboarding materials, it may be helpful to show different categories of requests that users may expect to work well, but that surprisingly don't, to help calibrate expectations."

Competencies

Finally, users' existing *competencies* inform how they use AIDEs. This relates to the idea of **shepherding** as an *acquired skill* (see 3.4.1) but also encompasses how general programming abilities impact one's readiness to use AIDEs.

Jayagopal et al. [20]: "Participants did not always know how best to specify their intent to the synthesizers they used. In particular [...] what kinds of specification were best for a given synthesizer."

Interview i12: *I think Copilot would help me a lot with programming assignments, but "only because I have [...] so much experience with it [...] if someone has, like, no coding experience, they won't be able to manipulate it" to give them what they want.*

Interview i15: "you still kind of need [...] coding experience [...] if I had this at the beginning of the semester, I would have been so confused."

3.5 Discussion

3.5.1 RQ1: How do novice programmers and AIDEs interact while programming under time pressure?

We found that novices' goals and values encouraged them to steer interactions with GitHub Copilot, yet they were also highly submissive to Copilot's direction. "Copilot" often acted more like a "pilot," leaving the human to second-in-command. The data showed mixed interaction quality and propensities toward the passive side of several thematic dimensions including bimodal interaction theory [3], leadership and intentionality, and the allocation of work within the team.

Planning

Barke et al. developed the bimodal theory of interaction to describe how novices interacted with GitHub Copilot either for **acceleration** or **exploration**. The **exploration** mode, they say, helps users to "*plan* their code actions", while the **acceleration** mode helps them to "*execute* their planned code actions" [3]. In our looser interpretation, **acceleration** would be better described as helping users to efficiently execute *some* coding action, not necessarily of their own choosing. Plenty of studies have described behaviors consistent with **acceleration** [3, 20, 27, 34, 40, 43, 51, 52] and **exploration** [3, 18, 23, 27, 34, 40, 43, 51, 52].

Participants in the present study also shared comments aligning with both modes, but they more often reported manifestations of **acceleration**. This can be explained by the **mindlessness** demonstrated by participants' lack of planning and intentionality. **Acceleration** allows code to be produced quickly and with little thought, but **exploration** provides more fruitful benefits such as enhanced planning through *consulting* [23] and overcoming the barriers of *getting started* [51] or *getting unstuck*. Further, a focus on speed leads to *rushing* that prioritizes timeliness over

understanding [34], overemphasizes salient UX features (*tunnel vision*) [20], and encourages shortsighted thinking (*myopia*) [40]. These forms of **mindlessness** can compound on existing tendencies toward approaching programming tasks with no strategy [27].

We argue that our participants used the **exploration** mode less because they were rarely developing plans of their own. Ability level is likely relevant here, as the original theory of Barke et al. was developed against data from renowned programming live streamers and participants with strong programming backgrounds [3]. Advanced programmers have additional **competencies** for planning and considering possible approaches they might use to complete a task. This aspect of **prior knowledge** allows them to take advantage of the powerful **exploration** mode with an AIDE rather than mindlessly taking suggestions to simply feel like they are progressing [20].

Authority

Our results suggest that a user's authority over an AI interaction is a precursor for effective planning. Our data showed that not only were participants **shepherding** [40] the AI, but the AI was **influencing** them in return. The prevalence of **shepherding** demonstrates that participants were active collaborators in the interaction. Prior work also provides strong evidence for this conclusion [3, 18, 20, 21, 23, 27, 34, 40, 43, 51, 52]. Though, delving into the types of shepherding participants used, *prompting* [3, 20, 21, 23, 27, 43], *adapting* [3, 18, 21, 23, 27, 40, 43, 51], and *backtracking* [3, 18, 23, 40, 51] were most prevalent. That is, participants were telling Copilot what they wanted and coping with its responses as opposed to leading the problem-solving effort. This sometimes led to *drifting* [3, 18, 40, 51] from one wrong solution to another, which, like *tinkering* [23, 43], we suspect happened more often than participants discussed. We labeled much of our data reminiscent of *drifting* instead as **shepherding**, **dissonance**, **mindlessness**, or **influencing** since seldom did participants provide a perfect match to our definition of *drifting*: a maladaptive and wasteful form of **shepherding**. Regardless of the naming, it is clear that AIDEs can disrupt the natural thinking process [52] by *polluting* [3] it with new ideas, *tempting* [40] users to take shortcuts, and squelching the uniqueness of one's own coding *style* [3]. Prather et al. found similar patterns, by which generative AI disrupted novices' metacognition by "misleading"

and "interrupting" them while they were trying to code [39]. For novices, these disruptions may outweigh benefits like exposure to technical *variety* [52] or the AI's potential for *teaching* them through its suggestions [43].

Notably, our analysis showed no examples of *decomposition* and almost no examples of *slow accepting* [40]. We reason about these as more adaptive forms of **shepherding**, in which a user displays authority by slowing down to think and plan their approach. It is fitting, then, that mindless and submissive novices with limited **prior knowledge** would be unlikely to engage in these practices. Several prior studies with participants of ranging ability levels reported examples of *decomposition* [3, 20, 21, 23, 43], but of these only [21] enforced time pressure (the most relaxed form of studies with time pressure [21, 34, 51]). Thus, when time is of the essence, *decomposition* may be the first form of **shepherding** to be abandoned. Unsurprisingly, *slow accepting* is also not an ideal strategy when a faster alternative is clearly available and preferable.

Finally, the data supporting **shepherding** as an *acquired skill* further emphasized the importance of *competencies* [20, 21, 40, 43] and *expectations* [3, 20, 21, 27, 40, 43, 52] for successful interactions with AIDEs. Studies have repeatedly supported that working proficiently with an AIDE is a nontrivial and challenging art [3, 18, 20, 21, 34, 43, 51]. Without sufficient **prior knowledge** entering an interaction, a novice is not ready to perform at their best in a team with AI. The original quantitative results from this study support this same conclusion [17].

Work Allocation

In addition to **outsourcing** [40] work to GitHub Copilot [3, 18, 21, 23, 27, 40, 43, 51, 52], participants also demonstrated **insourcing** of work, i.e. doing it themselves instead [3, 18, 20, 23, 27, 40, 43, 51, 52]. Participants' choices showed some promising leadership of Copilot through **insourcing** of pivotal aspects of software development like *problem solving*, *goal setting*, and *understanding* of code while **outsourcing** shallower tasks: *simple work* [3, 21, 23, 27, 43, 51, 52] and *expression* of ideas into programming language [3, 23, 27, 40, 43, 52]. However, these data are overshadowed by numerous manifestations of *over-reliance* [3, 23, 40, 43, 51, 52] and **insourcing**'s largest subtheme, *evaluating* [3, 18, 20, 23, 27, 40, 43, 52]. Assessing the quality

of AI-generated code is surely an adaptive strategy, but it is also a reactive one whereby the human user is responding to the activity of the AIDE as opposed to the reverse. Such a dynamic creates the conditions for mindless *drifting* and **influencing** to occur.

What is more, the top *evaluation* strategy used was *pattern matching* [3, 27], a faster alternative to *inspection* [3, 20, 40, 52]; rather than reviewing the details of code, a user simply skims code to find expected pieces of text [3]. Though often effective, this technique suffers from two major drawbacks. First, in the case of a *mirage* [3, 18, 43, 51], a human can believe that code is correct, missing small issues that are not readily apparent. Such oversights are also part of why humans experience *overload* when AIDE interfaces are too complex or verbose [3, 18, 20, 27, 34, 40, 43, 51, 52]. Secondly, a user's **prior knowledge** heavily restricts the space of possible outputs from the AIDE they deem acceptable; their *expectations*, *biases*, and *competencies* impact what they *expect* AI can or should do. For example, Section 3.4.3 describes how participants judged Copilot's behavior *against what they would do alone*. The obvious problem is that, as novices, what they would do alone might not necessarily be optimal. It follows that the ability to experience benefits from using an AIDE, such as interaction **harmony** [3, 18, 20, 21, 27, 34, 40, 43, 51, 52] or exposure to a *variety* [3, 40, 43, 51, 52] of programming constructs, is restricted through rejection of suggestions that could beget those benefits.

Rapport

Participants did, nevertheless, experience **harmony** with Copilot as in many prior studies [3, 18, 20, 21, 27, 34, 40, 43, 51, 52]. In general, the quality of their experiences came down to *alignment* [3, 18, 20, 40, 43, 51, 52] versus *misalignment* [3, 18, 20, 21, 23, 27, 34, 40, 43, 51, 52]. When Copilot offered *helpful* [3, 18, 40, 43, 51, 52] suggestions that provided good *clarity*, this made it *easier* [20] to program. The significantly lower effort and mental workload reported in the original quantitative results from this study further support this claim [17]. However, Copilot just as often provided *unhelpful* [18, 27] or *incorrect* [23, 27, 34, 43, 51] suggestions that caused *confusion* [20, 27, 40, 51, 52] and *overload* [3, 18, 20, 27, 34, 40, 43, 51, 52] if not *enabling* [40] mistakes of participants' own. In some cases, participants' *biases* and *expectations* played into

the quality of their interaction experiences. For example, *apprehension* [27, 40] about the future of AI was, in one case, fueled by dystopian imagery from popular culture. Enforcing conditions of time pressure, we pushed participants to their limits and uncovered the unfortunately two-sided nature of programming with AI. Forced to make real progress [20] with the AIDE, participants were less biased by the pleasantries and novelty [21] of casual AIDE interactions like those in other prior studies (e.g. [40, 43]). In contrast to our expectations from prior work, **harmony** and **dissonance** did not often manifest as *emotional* [3, 20, 21, 40, 51] in nature nor did participants engage in **humanizing** [18, 40, 43, 52] the AIDE quite like **objectivizing** it [18, 43, 52]. Again, the quantitative results corroborate this finding [17].

3.5.2 RQ2: What factors can explain how novices use AIDEs while programming under time pressure?

Time Pressure

Time pressure (a la *rushing*) added another layer to interaction dynamics. Expressions of **mindlessness** (e.g. *tunnel vision* and *myopia*) came from the apparent drive of a novice to take immediate action in pursuit of progress. Consider the programming of a basic iterative process. An advanced programmer might choose between a while loop, a for loop, a higher order function, or recursion. A novice, by comparison, might only know a standard for loop. Since the novice only knows one approach, the most efficient step they know to take is the one that feels most like progress: getting more code into the editor [7]. Planning efforts feel unproductive and wasteful to a novice when they think they know only one way forward. This exacerbates *over-reliance*, **mindlessness** and the power of **influencing**. When *polluting* occurs, it can be difficult for a novice to "dislodge" a mental model that is incorrect [38]. Further, because of *pattern matching* to a small set of *expectations*, novices have limited openness to AI's advice even as they are swayed so strongly by it. *Rushing* also encourages *drifting*; by mindlessly taking easy actions, novices miss the hidden value of *variety* and *helpful* suggestions that could be obtained by taking a little more time for **exploration**. If they lose a sense of how close they are to being done, they

might choose to ignore potentially useful suggestions that change their approach fundamentally [39]. Only when they find no way forward do they resort to *backtracking*, looking again for quick solutions.

Prior Knowledge

As discussed throughout Section 3.5.1 above, **prior knowledge** lays a key foundation for a novice's interaction with AI. Along with demonstrating the power of GitHub Copilot, the *training* [21] we gave participants showcased Copilot's fallibility, giving at least two participants *biases* [21, 43] against brazenly following its suggestions. *Competencies* and *expectations* further mediated interactions by determining what novices knew how to do with AI, how much leadership they could provide to it, and how they then reacted to what happened [3, 20, 21, 27, 40, 43, 52]. In Section 3.5.4 ahead, we discuss the potential of educational *training* to strategically build **prior knowledge** for users.

Values & Necessities

In line with prior studies [18, 21, 40, 52], participants shared extensively about the meaning of **responsibility** when working with AIDEs. They felt that collaborative *participation* [18, 40, 52] with the AI was both necessary and upstanding. One might suggest that the technical limitations of this particular AI materialization fueled this sentiment. Yet, amidst rapid technological progress, automation has, for decades, rendered humans, ironically, responsible for vigilant supervision and intervention after failures [2]. We argue that a combination of pragmatism and moral conviction will continue to justify human *participation* in AI teams even as technology continues to advance. Novices in the present study discussed *integrity* [40] and *accountability* as values relevant to using an AIDE and noted that *maturity* [21], *moderation*, and *context* [18] impact their appraisals of human-AIDE interactions. These findings lead to the conclusion that unfettered (no *moderation*) use of AI by novices (low *maturity*) in an educational *context* is morally disfavorable, even to novices themselves.

Within the experimental context, participants only somewhat lived out their values. While believing that **insourcing** of heady work like *problem solving*, *goal setting*, and *understanding* was important, these seldom manifested during the trials. Participants' reactive **insourcing** (see 3.5.1) suggests that they primarily did work out of necessity rather than desire. They mentioned that *control* [27, 51] and *fun* could motivate **insourcing**, but the *tempting* prospect of *easier* progress with Copilot often led to *outsourcing* instead, in some cases to a point of *over-reliance*. This result is consistent with the findings of Prather et al. who found that struggling novice programmers showed an "illusion of competence," by which their actions did not match their beliefs [39].

It was reasonable to assume, though, that typical values and motivations should not apply in this situation; participants were specifically instructed to complete as many tasks as possible and use the AI as much or as little as needed to maximize productivity. Evidently, participants felt sufficient **trust in AI** to rely on it heavily for this purpose unless given some reason not to. Herein lies a critical aspect of our study design that we believe muted the role of trust in participants' choices: participants had provided test cases against which to measure the AI's performance objectively. By comparison, Ross et al. did not give participants any running or debugging capabilities, requiring participants to build (subjective) **trust in AI** through communication or other means [43]. In other words, programmers act based on feelings only in the absence of sufficient informative logic [25].

3.5.3 RQ3: How does using an AIDE under time pressure affect a novice's thoughts and feelings?

Mild Reactions

In general, participants seemed largely unaffected by their interactions with AI. They experienced some of the best and the worst of what Copilot had to offer them (see 3.5.1). We saw this translate to their acknowledging its *inconsistency* of performance [43] and *not blaming* [18, 43, 52] it when **dissonance** occurred. Along with the *dispassion* that came through in their comments, many

participants referred to Copilot as a tool or *resource* rather than an intelligent agent. Others subscribed to a programming *teleology* in that they felt strongly about Copilot only based on the extent to which it helped them to complete tasks successfully [52]. Together, these findings led us to conclude that novices under time pressure tend toward **objectivizing** an AIDE rather than **humanizing** it as suggested by prior work [18, 40, 43, 52]. Participants did, however, sometimes exhibit **humanizing** by comparing Copilot to humans or applying traits to it like intelligence and understanding. We do believe that AIDEs with more anthropomorphic features, such as GILT [34] or newer versions of Copilot with Copilot Chat, can better encourage **humanizing**. Seeing AI in this way has the potential to help users better relate to and use AIDEs, but might also cause inappropriate *expectations* and behaviors [48].

As discussed in Section 3.5.4, the quality of a participant’s experience with Copilot was determined primarily by *alignment* of their thoughts and intentions to Copilot’s suggestions. A few participants felt frustration, enjoyment, or peace working with Copilot, but participants did not typically have *emotional* reactions to the quality of their interactions as expected based on prior work [3, 20, 21, 40, 51]. The quantitative results back up these findings. Copilot had non-significant effects to reduce frustration compared to the control condition [17]. That is, Copilot may have reduced some of the tension and frustration associated with programming, but introduced new issues, offsetting the reduction.

Self-Efficacy

Our data showed that an AIDE can have both negative and positive impacts on novices’ **self-efficacy**, but evidence of *low-self efficacy* [34, 40] was more prevalent than *high self-efficacy*. In line with the quantitative results [17], there was no overwhelming effect of GitHub Copilot to increase participants’ appraisal of their abilities, but they did build **self-efficacy** by honing the *acquired skill* of **shepherding** and observing its effect on their performance in a team with Copilot. Some also felt increased confidence to succeed in CS knowing they could use AIDEs as an additional learning resource. However, something not captured in the original results is the *self-blaming* novices did when noticing poor team performance as a result of poor **shepherding**.

Additionally, they did not always feel they could take credit for success with Copilot, describing it as unsatisfying or at least less satisfying than working alone. In a few cases they even experienced *emotional dissonance* in the form of *emptiness* and *shame* about accepting its help. The potential self-efficacy harms of AIDEs are especially alarming in light of the finding by Margulieux et al. that students who used AIDEs most during an introductory programming course had the lowest self-efficacy [30]. That is, those students whose self-efficacy is already low may face greater self-efficacy harm from AIDEs as compared to students with higher baseline self-efficacy.

3.5.4 Implications & Recommendations

Based on our findings, we recommend that educators prepare computer scientists to work in hybrid intelligence teams according to two pillar learning objectives: (1) maximizing useful output of the team, (2) preparing to intervene when needed.

Teach Effective Shepherding

Educators must prepare students for productivity-focused roles where they may be allowed, encouraged, or required to use AIDEs. Over decades, the software development industry has become increasingly motivated to produce software quickly and efficiently [45]. The ability of AIDEs to improve programming performance and empower less skilled programmers to produce useful code [17, 22] undoubtedly creates economic pressure for companies to adopt AIDEs urgently. CS education will need to adjust to these conditions with similar urgency [5, 24, 41]. Without specific instruction, the novices in the present study took time to adjust to using Copilot and ultimately gravitated most toward mindless and submissive strategies. The quality of LLM-generated code depends heavily on the quality of the inputs provided [11], and, as in [21], our results showed that users can discover this through experience. UX researchers are working to make AIDE interaction more intuitive (e.g., [32]), but students still need to learn and practice this skill. Additionally, our results suggest that doing so can provide a new way for students to build **self-efficacy**. Notably, our participants did not naturally tend toward problem *decomposition*, a

critical step in solving real-world problems and the focus of a full chapter in Porter and Zingaro’s AI-assisted programming textbook [37]. Pedagogical approaches akin to “Prompt Problems” [12] can help to isolate different types of **shepherding** so that students can avoid overusing some over others. For example, *decomposition* could be targeted by having students rewrite correct prompts to better specify subgoals or *adapting* could be targeted by having students modify AI-generated code.

Ensuring that the code students learn to produce with AIDEs is *useful* also requires that students learn to use AIDEs for a broad range of tasks. When Copilot could not produce a full solution for a participant, they would generally do high-level tasks (e.g. *problem solving*) and *evaluation* of code themselves, while using Copilot for shallower tasks like *expression* of ideas into code. Students need instruction on how to utilize Copilot for tasks they otherwise only know how to do manually (without fully relying on Copilot for an entire task). They can then recognize a wider range of opportunities to apply AI assistance (e.g. testing, documentation, algorithm development) and do less work, while remaining engaged.

Enable Human Intervention

Educators must teach students why and how they should participate in software development in the age of AI. If strong human direction led automation to flawless performance, it would be the only necessary competency. However, AIDEs can produce code that fails to meet functional requirements (e.g. correctness, reliability, and scalability) or non-functional requirements (e.g. originality, style, and readability). Further, programmers may lack access to AIDEs due to temporary outages or permanent financial or legal constraints. For these reasons, well-rounded programmers need to be able to supervise AIDEs and code without them if needed. The novices in this study relied heavily on GitHub Copilot to accelerate their programming, exacerbating aimless programming behaviors and harming **self-efficacy**. When reflecting on the experience, many identified how fast pacing and *over-reliance* made it challenging for them to fully scrutinize and learn from AI-generated code. We suggest three instructional thrusts to counteract these tendencies. First and foremost, educators can introduce students to the latest AIDE technology,

how it works, how to access and use it, and what its implications are. Second, educators can prohibit the use of AIDEs for some assignments. This will incentivize students to learn from AI when they can use it, rather than to simply maximize productivity. It will also allow them to build **self-efficacy** by recognizing their abilities independent of assistance. Third and finally, educators can provide specific instruction on interventions to AI-generated code. For example, the approach in Porter and Zingaro’s textbook [37] provides introductions to reading, testing, and debugging AI-generated Python code. Our participants’ propensity toward the *pattern matching* evaluation technique points especially to a need for teaching reading and testing competencies. We saw some evidence that our training on Copilot’s fallibility made an impression on our participants, but they only truly learned to cope with this from firsthand experience. For this reason, we suggest that students be intentionally exposed to erroneous AI outputs so that they can learn to slow down, approach large suggestions with caution, and inspect code carefully.

3.6 Limitations

Because our data was collected from the participants of our prior study, the limitations discussed in the original paper [17] also apply here. Our instructions to participants to prioritize productivity present an additional threat to ecological validity, as real students typically have some motivation to learn. However, we hope that this design choice sheds light on how students might compromise on their values in service of meeting a deadline or acing an assignment in a worst-case scenario. Furthermore, our qualitative approach introduces several potential threats to validity. Research biases certainly shaped the outcome of our investigation, as is expected when drawing on reflexive qualitative methods [8]. To mitigate the effects of our biases, we attempted to avoid loaded or leading wording [4] in interview questions and performed member checking [28] of participants’ responses. We also drew on analysis and internal critique amongst three researchers when formulating the early versions of our framework and selecting meaningful quotes. While the organization and emphasis of the ideas in the framework are highly subjective, we demonstrate the validity of the ideas through a strong basis in existing literature and the use of direct

quoting. Our IRB does not permit public release of our raw data but does allow us to share it with other researchers upon request. Finally, because of differences in personality, language abilities, interview length, expressiveness, etc. we were concerned about overrepresenting certain participants in our analysis. To counteract selective reporting, we attempted to balance the use of direct quotes across participants. For transparency, readers can review our selection statistics in Appendix C.

3.7 Future Work

Future work can focus on testing our framework and exploring the efficacy of instructional approaches to carry out our recommendations. Because our framework was co-constructed out of both prior work and our dataset, different ideas might be emphasized when considering different groups of students or different experimental conditions. For example, it remains to be seen whether novices who are incentivized to learn would do so when using an AIDE under time pressure. It is also unclear how specific our participants' attitudes are to their unique educational and cultural backgrounds. Regarding pedagogy, we are only aware of one [37] formal approach to teaching AI-assisted programming. It is intended as an introduction to programming that encourages using AIDEs from day one. Further effort should be made to craft curricula that utilize AIDEs in different ways and for students of different ability levels.

3.8 Conclusion

In this article, we synthesized a model for human-AIDE interaction from existing literature and analyzed qualitative data from seventeen semi-structured interviews to assess and extend the model. We validated numerous themes from prior work including **acceleration**, **shepherding**, **exploration**, **mirroring**, **dissonance**, **harmony**, **outsourcing**, and **humanizing**. We also formalized several other themes including **mindlessness**, **influencing**, **insourcing**, **responsibility**, and **objectivizing**. Our results stress the importance of novices' value **appraisals** and **prior**

knowledge for determining outcomes, but also show that novices often act against their values when placed under time pressure. We acknowledge the subjectivity and limitations of our framework and encourage other researchers to further assess our findings. Our educational recommendations are for instructors to include AIDEs in their curricula through both performance-centered AIDE-permitted assignments and solo assignments for building independence and AIDE intervention skills. Future work can explore applied pedagogical approaches for hybrid intelligence teaming and examine the effects of learning-based incentives compared to performance-based incentives on novices' interactions with AIDEs.

Acknowledgments

This work was supported by the UVA Distinguished Fellowship, NSF NRT (1829004) & GRFP (1842490), and the Virginia CCI CVN (GR102900). We thank Claire Cebula and Ashraf Ibrahim for their assistance with analyzing this data and developing our framework.

References

- [1] Saleema Amershi et al. 2019. Guidelines for human-AI interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19: CHI Conference on Human Factors in Computing Systems. ACM, Glasgow Scotland UK, (May 2, 2019), 1–13. ISBN: 978-1-4503-5970-2. doi:10.1145/3290605.3300233.
- [2] Lisanne Bainbridge. 1983. Ironies of automation. In *Analysis, design and evaluation of man-machine systems*. Elsevier, 129–135.
- [3] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: how programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7, 85–111, OOPSLA1. Number: OOPSLA1.
- [4] Kathy Baxter, Catherine Courage, and Kelly Caine. 2015. *Understanding your users: a practical guide to user research methods*. Morgan Kaufmann. Retrieved Aug. 28, 2024 from <https://books.google.com/books?hl=en&lr=&id=l9-cBAAQBAJ&oi=fnd&pg=PP1&dq=Understanding+your+users:+A+practical+guide+to+user+research+methods&ots=dEIEZAXoRZ&sig=4ckI45L5qds4FShqQmSV-u-KsFO>.

- [5] 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*, 500–506.
- [6] H. Russell Bernard, Amber Wutich, and Gery W. Ryan. 2016. *Analyzing qualitative data: Systematic approaches*. SAGE publications. Retrieved July 26, 2024 from <https://books.google.com/books?hl=en&lr=&id=yAi1DAAAQBAJ&oi=fnd&pg=PP1&dq=Analyzing+qualitative+data:+Systematic+approaches.&ots=SzmVQwMVEi&sig=A61SBZ49PHw1sQcIoRsyI1cqSXo>.
- [7] 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. *Commun. ACM*, 66, 6, (May 24, 2023), 56–62. doi:10.1145/3589996.
- [8] Virginia Braun and Victoria Clarke. 2022. Conceptual and design thinking for thematic analysis. *Qualitative psychology*, 9, 1, 3. Retrieved July 12, 2024 from <https://psycnet.apa.org/fulltext/2021-45248-001.html>.
- [9] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3, 2, (Jan. 2006), 77–101. doi:10.1191/1478088706qp063oa.
- [10] Ruijia Cheng, Ruotong Wang, Thomas Zimmermann, and Denae Ford. 2024. “it would work for me too”: how online communities shape software developers’ trust in AI-powered code generation tools. *ACM Trans. Interact. Intell. Syst.*, 14, 2, (May 15, 2024), 11:1–11:39. doi:10.1145/3651990.
- [11] 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*. SIGCSE 2023: The 54th ACM Technical Symposium on Computer Science Education. ACM, Toronto ON Canada, (Mar. 2, 2023), 1136–1142. ISBN: 978-1-4503-9431-4. doi:10.1145/3545945.3569823.
- [12] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt problems: a new programming exercise for the generative AI era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 296–302. ISBN: 979-8-4007-0423-9. doi:10.1145/3626252.3630909.
- [13] Benedict Du Boulay, Tim O’Shea, and John Monk. 1981. The black box inside the glass box: presenting computing concepts to novices. *International Journal of man-machine studies*, 14, 3, 237–249. Retrieved Sept. 9, 2024 from <https://www.sciencedirect.com/science/article/pii/S0020737381800569>.

- [14] Neil A. Ernst and Gabriele Bavota. 2022. AI-driven development is here: should you worry? *IEEE Software*, 39, 2, (Mar. 2022), 106–110. Conference Name: IEEE Software. doi:10.1109/MS.2021.3133805.
- [15] Dr Nigel G. G. Fielding, Nigel Fielding Raymond M. Lee, and Professor Raymond M. Lee. 1998. *Computer Analysis and Qualitative Research*. Google-Books-ID: J5ZX9qbCylgC. SAGE, (June 8, 1998). 218 pp. ISBN: 978-1-4462-2815-9.
- [16] 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 (ACE '23)*. Association for Computing Machinery, New York, NY, USA, (Jan. 30, 2023), 97–104. ISBN: 978-1-4503-9941-8. doi:10.1145/3576123.3576134.
- [17] Nicholas Gardella, Raymond Pettit, and Sara L. Riggs. 2024. Performance, workload, emotion, and self-efficacy of novice programmers using AI code generation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2024: Innovation and Technology in Computer Science Education*. ACM, Milan Italy, (July 3, 2024), 290–296. ISBN: 979-8-4007-0600-4. doi:10.1145/3649217.3653615.
- [18] Gian Luca Scoccia. 2023. Exploring early adopters' perceptions of ChatGPT as a code generation tool, (Sept. 11, 2023). MAG ID: 4388214696. doi:10.1109/asew60602.2023.00016.
- [19] Barney G. Glaser and Anselm L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research. Observations*. Aldine Pub. Co, Chicago.
- [20] 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*, 1–15.
- [21] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J. Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–19.
- [22] 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, (Apr. 19, 2023), 1–23. ISBN: 978-1-4503-9421-5. doi:10.1145/3544548.3580919.

- [23] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research* (Koli Calling '23). Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–12. ISBN: 979-8-4007-1653-9. doi:10.1145/3631802.3631806.
- [24] Sam Lau and Philip 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 2023: ACM Conference on International Computing Education Research. ACM, Chicago IL USA, (Aug. 7, 2023), 106–121. ISBN: 978-1-4503-9976-0. doi:10.1145/3568813.3600138.
- [25] John D. Lee and Katrina A. See. 2004. Trust in automation: designing for appropriate reliance. *Human factors*, 46, 1, 50–80. Publisher: SAGE Publications Sage UK: London, England.
- [26] Michael J. Lee and Amy J. Ko. 2011. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*. ICER '11: International Computing Education Research Workshop. ACM, Providence Rhode Island USA, (Aug. 8, 2011), 109–116. ISBN: 978-1-4503-0829-8. doi:10.1145/2016911.2016934.
- [27] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A large-scale survey on the usability of AI programming assistants: successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (ICSE '24). Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–13. ISBN: 979-8-4007-0217-4. doi:10.1145/3597503.3608128.
- [28] Yvonna S. Lincoln and Egon G. Guba. 1985. *Naturalistic inquiry*. Vol. 75. Sage. Retrieved Aug. 28, 2024 from <https://books.google.com/books?hl=en&lr=&id=2oA9aWlNeooC&oi=fnd&pg=PA7&dq=lincoln+and+guba+naturalistic&ots=0v1xYcVfm&sig=0g8kKlMXyrx54UVh08HGQdrXLZI>.
- [29] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J. Cai. 2020. Novice-AI music co-creation via AI-steering tools for deep generative models. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20: CHI Conference on Human Factors in Computing Systems. ACM, Honolulu HI USA, (Apr. 21, 2020), 1–13. ISBN: 978-1-4503-6708-0. doi:10.1145/3313831.3376739.
- [30] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-regulation, self-efficacy, and fear of failure interactions with how novices use LLMs to solve programming problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2024: Innovation and Technology in Computer

Science Education. ACM, Milan Italy, (July 3, 2024), 276–282. ISBN: 979-8-4007-0600-4.
doi:10.1145/3649217.3653621.

- [31] Stephanie M. Merritt, Heather Heimbaugh, Jennifer LaChapell, and Deborah Lee. 2013. I trust it, but i don't know why: effects of implicit attitudes toward automation on trust in an automated system. *Human factors*, 55, 3, 520–534. Number: 3 Publisher: Sage Publications Sage CA: Los Angeles, CA.
- [32] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Ben 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 MAG ID: 4365601419. (Apr. 19, 2023).
doi:10.1145/3544548.3580817.
- [33] Matthew B. Miles and A. Michael Huberman. 1994. *Qualitative data analysis: An expanded sourcebook*. sage. Retrieved July 26, 2024 from
https://books.google.com/books?hl=en&lr=&id=U41U_-wJ5QEC&oi=fnd&pg=PR12&dq=Qualitative+data+analysis:+An+expanded+sourcebook&ots=kGUB_EMRWR&sig=Lly8iJD8cZSpIL_hkME2aeTtlvk.
- [34] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24: IEEE/ACM 46th International Conference on Software Engineering. ACM, Lisbon Portugal, (Apr. 12, 2024), 1–13. ISBN: 979-8-4007-0217-4. doi:10.1145/3597503.3639187.
- [35] Emily Namey, Greg Guest, Lucy Thairu, and Laura Johnson. 2008. Data reduction techniques for large qualitative data sets. *Handbook for team-based qualitative research*, 2, 1, 137–161. Retrieved Aug. 28, 2024 from <https://www.researchgate.net/profile/Nageswara-Posinasetti/post/How-do-you-deal-with-rich-too-much-data-for-a-qualitative-research-study/attachment/59d62f87c49f478072ea000a/AS%3A273588442075143%401442239975624/download/Data+Reduction+Techniques.pdf>.
- [36] Ipek Ozkaya. 2023. The next frontier in software development: AI-augmented software development processes. *IEEE Software*, 40, 4, (July 2023), 4–9. Conference Name: IEEE Software.
doi:10.1109/MS.2023.3278056.
- [37] Leo Porter and Daniel Zingaro. 2023. *Learn AI-Assisted Python Programming With GitHub Copilot and ChatGPT*. Manning, (Nov. 21, 2023). ISBN: 978-1-63343-778-4.
<https://www.manning.com/books/learn-ai-assisted-python-programming>.
- [38] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive difficulties faced by novice programmers in automated assessment tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ICER '18: International

- Computing Education Research Conference. ACM, Espoo Finland, (Aug. 8, 2018), 41–50. ISBN: 978-1-4503-5628-2. doi:10.1145/3230977.3230981.
- [39] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: the benefits and harms of generative AI for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*. ICER 2024: ACM Conference on International Computing Education Research. ACM, Melbourne VIC Australia, (Aug. 12, 2024), 469–486. ISBN: 979-8-4007-0475-8. doi:10.1145/3632620.3671116.
- [40] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. “it’s weird that it knows what i want”: usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 29, 2024), 1–31. doi:10.1145/3617367.
- [41] James Prather et al. 2023. The robots are here: navigating the generative AI revolution in computing education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITICSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, (Dec. 28, 2023), 108–159. ISBN: 979-8-4007-0405-5. doi:10.1145/3623762.3633499.
- [42] Stephen Rice, David Keller, Gayle Hunt, and David Trafimow. 2009. Automation dependency under time pressure. *2009 International Symposium on Aviation Psychology*, (Jan. 1, 2009), 611–616. https://corescholar.libraries.wright.edu/isap_2009/14.
- [43] 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. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. IUI '23: 28th International Conference on Intelligent User Interfaces. ACM, Sydney NSW Australia, (Mar. 27, 2023), 491–514. ISBN: 979-8-4007-0106-1. doi:10.1145/3581641.3584037.
- [44] Herbert J. Rubin and Irene S. Rubin. 2011. *Qualitative interviewing: The art of hearing data*. sage. Retrieved Aug. 28, 2024 from <https://methods.sagepub.com/book/qualitative-interviewing>.
- [45] Caitlin Sadowski and Thomas Zimmermann. 2019. *Rethinking productivity in software engineering*. Springer Nature. Retrieved Sept. 9, 2024 from <https://library.oapen.org/handle/20.500.12657/22839>.
- [46] Sage. 2022. Common challenges in thematic analysis and how to avoid them with virginia braun & victoria clarke. (Aug. 11, 2022). Retrieved Sept. 1, 2024 from <https://www.youtube.com/watch?v=tpWLsckpM78>.
- [47] Johnny Saldaña. 2021. The coding manual for qualitative researchers. Retrieved July 25, 2024 from <https://www.torrossa.com/gs/resourceProxy?an=5018667&publisher=FZ7200>.

- [48] Isabella Seeber et al. 2020. Machines as teammates: a research agenda on AI in team collaboration. *Information & management*, 57, 2, 103174. Retrieved Aug. 30, 2024 from <https://www.sciencedirect.com/science/article/pii/S0378720619303337>.
- [49] Anselm L. Strauss and Juliet M. Corbin. 1990. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Sage Publications, Newbury Park, Calif. ISBN: 978-0-8039-3250-0 0-8039-3250-2 978-0-8039-3251-7 0-8039-3251-0.
- [50] Casey Tunstall, Stephen Rice, Rian Mehta, Victoria Dunbar, and Korhan Oyman. 2014. Time pressure has limited benefits for human-automation performance. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 58, 1, (Sept. 1, 2014), 1043–1046. doi:10.1177/1541931214581218.
- [51] 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 (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, (Apr. 28, 2022), 1–7. ISBN: 978-1-4503-9156-6. doi:10.1145/3491101.3519665.
- [52] Justin D. Weisz et al. 2021. Perfection not required? human-AI partnerships in code translation. In MAG ID: 3154248444. (Apr. 14, 2021), 402–412. doi:10.1145/3397481.3450656.
- [53] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14: 18th International Conference on Evaluation and Assessment in Software Engineering. ACM, London England United Kingdom, (May 13, 2014), 1–10. ISBN: 978-1-4503-2476-2. doi:10.1145/2601248.2601268.
- [54] Jed R. Wood and Larry E. Wood. 2008. Card sorting: current practices and beyond. *Journal of Usability Studies*, 4, 1, 1–6. Retrieved July 30, 2024 from https://uxpajournal.org/wp-content/uploads/sites/7/pdf/JUS_Wood_Nov2008.pdf.

Chapter 4

Relationships Between Trust, Compliance, and Performance for Novice Programmers Using AI Code Generation

Chapter Preamble

After quantitatively evaluating Copilot for novices in Chapter 2 and describing interactions qualitatively in Chapter 3, open questions remained about the role of trust in participants' usage of Copilot. This chapter builds on the limited findings in Section 3.4.4 of Chapter 3 by attempting to explain the chronological relationships between trust and AIDE compliance, between AIDE compliance and performance, and between performance and trust. The present investigation brings together a large body of past trust research in the realm of human factors with the more nascent study of human-AIDE interaction. This chapter completes the objectives of Aim 1 to produce four-

Authorship, Reuse, and Licensing Statement This chapter was coauthored with Sara L. Riggs  and Matthew Bolton , who provided explicit consent for the inclusion of this chapter in this dissertation. The study explores a dataset that overlaps with Chapter 2 because performance data from Cohort i was used in both analyses for different purposes. Neither performance data from Cohort j nor trust and compliance data from both cohorts have been previously published. The content from this chapter may appear in a future publication in *Human Factors: The Journal of the Human Factors and Ergonomics Society* (see Preprint for Chapter 4 in Appendix E). Otherwise, the fallback license for this chapter is listed on the copyright page for this dissertation. *Human Factors* currently has authors retain copyright and grant an Exclusive License to Publish (ELP) to Sage Publications. Sage allows for liberal reuse of publication content per the following section of the *Sage Green Open Access* policy, retrieved on March 18, 2026: "You may share the Original Submission or Accepted Manuscript at any time after your paper is accepted and in any format."

dational evaluations and explanations of novice-AIDE interaction. The following chapter begins Aim 2, moving from the narrower study of human-AI interactions to a study of broader implications centered around the interplay of AIDEs with computing identity and educational persistence for students at an HBCU.

Abstract

Objective

To explore how novice programmers' trust in Artificial Intelligence-driven Development Environments (AIDEs) relates to their coding performance and AI compliance while programming under time pressure.

Background

Computer programming has undergone rapid upheaval due to state-of-the-art AIDEs, which provide clever automation for many aspects of software development. A longstanding interest of researchers of automation more generally has been the attitude of trust. Decades of research seek to explain how influencing trust can help to achieve desirable outcomes in different domains, but very limited work has provided similar focus on trust in AIDEs.

Method

We collected subjective measures of trust along with objective measures of performance and AIDE compliance from a diverse group of 27 novice programmers between two study locations.

Results

Our results corroborated traditional understandings of how trust changes through experiences. However, we did not find a relationship between trust and subsequent compliance during pro-

programming tasks. Greater compliance was associated with strong performance, and strong performance led to greater subsequent trust.

Conclusion

Our findings raise new questions about the utility of trust in the context of interacting with AIDEs and generative AI. We call for further research into the effect of trust on compliance to recommendations from imperfect AI.

Application

This work can inform the design of training and educational content for generative AI use within and beyond software development. Instructional designers should consider risks of AI misuse and disuse and focus on promoting desirable interaction outcomes, regardless of trust's connection to them.

Précis

A study of 27 novice programmers using the GitHub Copilot AI-driven Development Environment explored correlations between pre-interaction trust, AI compliance, programming performance, and post-interaction trust. The results could not show a relationship from trust to compliance but did show positive relationships from compliance to performance and performance back to trust.

4.1 Introduction

Artificial Intelligence (AI) technology can be very effective within software development, a field which drives technological progress in a myriad of application domains. AI-driven Development Environments (AIDEs) can help to automate aspects of coding such as typing, translating, problem-solving, testing, and debugging. AIDEs typically offer chat messaging, interactive text

editing, or both. Programmers can use AIDEs as addons to their existing Integrated Development Environments to generate new text, edit existing text, learn more about a topic, or take an action (e.g., run a terminal command). AIDEs can produce outputs that are fast, highly effective, and well-liked, but can also produce incorrect or unsafe code [15, 49, 64, 67]. As such, it is important to study how programmers can use AIDEs to boost their productivity and well-being without compromising the safety of anyone their software will impact [16, 29, 31, 60]. Unfortunately, it is not yet possible to ensure that AIDEs will effectively balance these goals [31]. Instead, human programmers must decide whether and to what extent they should rely on AIDEs. According to the human-automation interaction literature, this decision is largely based on a user's *trust* in an automated system [16, 29, 47]. With this connection, ensuring trust aligns closely with reliability should lead to ideal reliance.

This work focuses on relationships between trust, compliance, and performance in the context of novice programmers using AIDEs. Novices, as compared to intermediate and advanced programmers, have limited programming expertise to bias their attitudes, intentions, and behaviors. They are ideal candidates for this work because their higher openness increases the likelihood of detecting any true relationships that might exist between trust, compliance, and performance. These three constructs have been central to several theoretical models of human-automation interaction [16, 29, 33], but have not been studied extensively in programmer-AI interactions. These constructs are important because they are measurable, interconnected, and either intrinsically desirable or responsive to intervention. Lee and See [29] can be credited for popularizing the notion that trust, compliance, and performance interact cyclically, informing many later models [56]. Performance is desirable because it is the purpose of the automation to begin with. Compliance and reliance can help optimize the automation's effect on performance. Finally, interventions can target trust as an avenue for optimizing compliance and thus performance. However, the utility of trust is limited if these connections break down. Therefore, the goal of this work is to evaluate the relationships between novice programmers' (a) trust in an AI code generator (b) compliance with the AI's suggestions, and (c) programming performance.

4.1.1 Trust, Compliance, and Performance

Given the nascence of AIDEs, there is limited existing work about how trust, compliance, and performance are related for programmers using AI. However, the broader knowledge base around human-automation interaction supports a cyclic progression between these three constructs. Trust is thought to inform compliance, which, based on reliability, impacts performance, and trust is then adjusted based on performance [16, 29, 33]. Figure 4.1 captures this expected cycle, which was crystallized by [29].

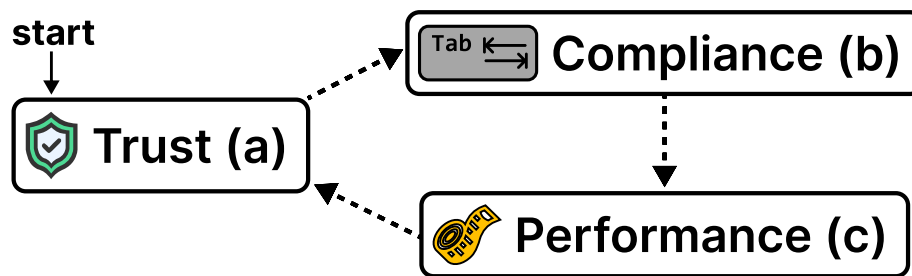


Figure 4.1: Expected cycle of trust, compliance, and performance from automation trust literature.

Note. Users start with an initial level of (a) trust in the AIDE. Next, trust informs (b) how much users comply with AIDE suggestions. When the AIDE is very reliable for the task at hand, this compliance impacts (c) performance. Depending on how the team performs, the user can reevaluate their (a) trust.

Note. Users start with an initial level of (a) trust in the AIDE. Next, trust informs (b) how much users comply with AIDE suggestions. When the AIDE is very reliable for the task at hand, this compliance impacts (c) performance. Depending on how the team performs, the user can reevaluate their (a) trust.

In addition to theoretical evidence, empirical studies support that trust can predict compliance and be predicted by performance. Quantitative evidence for trust predicting compliance comes from the automation trust literature [10, 40, 48, 68], but many qualitative studies of AIDEs have suggested this connection as well [23, 52, 57, 64]. As for performance predicting trust, quantitative evidence is also from the automation literature, but AIDE-related studies have supported this

connection, too [5, 21]. The following subsections further detail the overlap between empirical work and the theoretical underpinnings of a trust-compliance-performance cycle.

Trust

Human *trust* in automation (Figure 4.1a) is generally considered an attitude one actor has toward relying on another whose reliability is not fully knowable [1, 29, 33]. Trust varies in its generality—i.e. in whom or under what circumstances does one trust [16, 27, 32, 36, 40]. Here we focus on one’s specific trust in an AIDE at several points in time. After the trustor becomes familiar with the trustee, but before any substantial interactions, they can provide a baseline level of specific trust [3, 16, 40]. Thereafter, specific trust is “history-based” [27, 28, 40] because accumulated interactions can be taken into consideration.

Prior work has measured trust in AIDEs, but only once per participant. Amoozadeh et al. [2] found that students in an earlier CS course trusted AIDEs more than those in a later course at the same school. Similarly, Perry et al. found a trend for participants who wrote more secure code to trust an AIDE less than those who wrote less secure code [52]. These trends may be attributable to automation bias, which causes users to over-value salient automated cues, even when more reliable information is available [43]. Novices, especially, may over-trust AIDEs by naively comparing their suggestions to more reliable feedback mechanisms like compiler messages [54]. Novices have also been found to trust AIDEs more for code comprehension than for code completion, perhaps because of better trust affordances [62] or because of less perceived personal risk [43]. We are not aware of any studies taking repeated measures of trust in AIDEs, nor testing its effect on compliance.

Compliance

Compliance refers to acting on the advice of automation [11]. Specific trust is thought to influence humans’ *compliance* and *reliance* on automation [29], corresponding to the arrow between Figure 4.1a and Figure 4.1b; though, recent work questions the strength of this connection [48, 53].

Colloquially, terminology is used more loosely, but *reliance* is strictly defined as the choice *not to act* in the *absence* of an automation alarm [11, 29]. Therefore, the term *compliance* (Figure 4.1b) better describes depending on an AIDE by accepting suggested code, edits, or actions [11]. Empirical work supports a small positive relationship between pre-interaction trust in automation and compliance within the interaction [48]. Studies have found this connection in simulations of military decision support [10], X-ray luggage screening tasks [40], and multitask driving scenarios [68]. Though similar quantitative evidence is lacking in the human-AIDE interaction context, several qualitative studies have suggested that trust in an AIDE can influence compliance [23, 52, 57, 64].

On the other hand, factors such as high workload [10, 63] and low operator self-confidence [28] can also lead to greater compliance and reliance. When a user is confident to rely on their own abilities, they have a lesser need to use automation [12]. In the previously mentioned study about AIDEs and code security, participants with more prior security experience tended to use the AI-generated code less [52]. It is not clear whether they did so because of low trust in AI or simply high self-confidence.

The cause of AIDE compliance matters because numerous studies have claimed to show evidence of programmers over-relying on AIDEs [25, 54, 57, 62, 64]. Typically, over-reliance would be considered "misuse" [46] of automation—a "poor calibration in which trust exceeds system capabilities" [29]. This aligns with studies observing participants accepting wrong or insecure code from an AIDE [52, 57]. In contrast, other observations have been labeled over-reliance because of the risk, not the presence, of a bad outcome. For example, researchers often consider accepting large amounts of AI generated code without engaging with it (i.e., carefully reading or editing it) to be over-reliance [25, 62, 64]. However, if the code perfectly satisfies its intended need, the trust literature might consider such behavior to be appropriate reliance. One way to measure a bad outcome objectively is programming performance.

Performance

Performance is how well the human-automation team collectively performs the task at hand. In software development, it can be measured in many ways (e.g. readability, time complexity, or reliability), but for novices, the most relevant measures assess basic software functionality under time constraints [34]. For example, novice programming performance can be quantified as the duration required to produce a program that passes an objective set of tests, or the number of tests passed within a specified period.

In general, automation compliance improves performance when the automation is reliable for the task or degrades it otherwise [16, 29]. AIDEs are very reliable for the types of simple programming tasks in which novices engage [7, 14]. Novices also perform better with AIDE assistance than alone, so compliance should be positively related to performance, at least to some limit [13, 15, 24]. This corresponds to the arrow between Figure 4.1b and Figure 4.1c.

Performance can be used to adjust subsequent specific trust [29]. The performance of the automation itself has been positively associated with trust in simulations of industrial automation [28], military decision support [12], and multitask driving scenarios [68]. Extending these findings to the net performance of the human and the automation together, Merritt and Ilgen also found a direct relationship from performance to specific trust [40]. Unfortunately, no studies have validated this link for the AIDE context. However, at least two studies cite the perception of suggestion quality as an important factor for the development of trust in an AI code generator [5, 21]. This suggests that performing well with an AIDE would increase a user's trust in it provided that they notice how well they are performing [16]. This link corresponds to the arrow from Figure 4.1c back to Figure 4.1a, completing a cycle that can repeat over time.

4.1.2 Research Questions & Contributions

While the evolving knowledge base on human-AIDE interaction acknowledges the relevance of trust, compliance, and performance, there is a major gap in its ability to describe their interrelationships empirically. Most quantitative evidence to date comes from studies with other types of

automation. Our contribution is to empirically evaluate the cyclical trust-compliance-performance theory popularized by [29] in the context of human-AIDE interactions. To this end, we explore the following research questions in this study:

- **RQ1:** Does specific trust in an AIDE predict subsequent compliance (Figure 4.1, a to b)?
- **RQ2:** Does compliance predict performance when coding with an AIDE (Figure 4.1, b to c)?
- **RQ3:** Does programming performance predict subsequent specific trust in an AIDE (Figure 4.1, c to a)?

To answer these research questions, we planned to test three hypotheses using one regression coefficient test for each research question. First, for RQ1, we expected a positive relationship from specific trust (Figure 4.1a) to compliance (Figure 4.1b) [10, 40, 48, 68]. Due to the AIDE's high reliability on simple programming tasks like the ones we use [7, 14], we also expected a positive relationship between compliance (Figure 4.1b) and performance (Figure 4.1c) for RQ2 [16, 26, 29, 35]. Finally, we expected a positive relationship from performance back to specific trust (Figure 4.1a, post-interaction) for RQ3 [12, 28, 29, 40, 68].

4.2 Method

Data was collected at University of Virginia (UVA) in Spring 2023 and Virginia State University (VSU) in Spring 2024. Performance data from UVA was previously reported for a separate analysis [15]. This research complied with the American Psychological Association Code of Ethics and was approved by the Institutional Review Board at UVA for participants recruited there and at both universities for participants recruited at VSU. Informed consent was obtained from each participant.

4.2.1 Participants

In total, 27 participants were recruited by convenience as volunteers (16 female; 10 male; 1 unknown; 10 Black; 8 Asian; 3 Asian & White; 4 White; 1 Latinx & Black; 1 Latinx & White). Only three participants had any prior experience with coding-specific AIDEs, but many (24) had tried general-purpose AI chatbots. Seventeen UVA students (11 female; 5 male; 1 unknown; 8 Asian; 3 Asian & White; 4 White; 1 Latinx & Black; 1 Latinx & White) were recruited from a 1000-level Introduction to Programming (CS1) course. Ten VSU students (5 female; 5 male; 10 Black) were recruited from a 1000-level Introduction to Programming (CS2) course. By consulting syllabi and faculty at both universities, we determined that students in this second-semester intro course would have similar programming abilities to the participants from UVA. Nonetheless, we explicitly accounted for site-to-site variation throughout the analysis.

4.2.2 Experimental Setup

Participants at both sites programmed in the popular Visual Studio Code (VS Code) Integrated Development Environment on an Apple iMac with Python 3 and the GitHub Copilot IDE extension. At UVA, participants used GitHub Copilot circa 2023 (v1.79), while at VSU, participants used GitHub Copilot circa 2024 (v1.151) with GitHub Copilot Chat v0.11.1. This change kept VSU participants up to date with the newest supported versions available at the time and ensured that they did not encounter backwards compatibility issues with outdated remote services. Both versions provided inline suggestions and a multiple suggestions panel, but the newer version also had conversational options, context direction tools, and an interactive editor. Custom VS Code extensions provided task access, submission capabilities, and tracking of participants' interactions with Copilot. The interface showed a countdown timer, a PDF with task instructions, and a Python editor containing an unimplemented function. Participants could run and test their code using either the terminal or point-and-click options. They could advance to the next task in a trial only once their solution for the current task passed all the tests provided in the dataset—HumanEval (<https://github.com/openai/human-eval>; [7]).

4.2.3 Procedure

Participants at both sites were guided through the same procedure. They provided informed consent electronically and were given an overview of the study and an opportunity to ask questions. A series of training videos was presented to familiarize participants with the programming environment, the task submission tool, the functionality of GitHub Copilot, and the risks and benefits of using it. Because Copilot's functionality was expanded and the study software was updated slightly, participants at VSU were shown updated training videos on these topics to account for these updates. Participants at both sites viewed the same video about the risks and benefits of Copilot. While participants were shown different ways of using Copilot, they were not encouraged or discouraged from using any particular strategy. After each video, participants demonstrated their understanding either verbally or by completing the actions shown in the video.

As part of a larger study design, participants completed four Python programming blocks—two alone with standard IntelliSense autocomplete and two with GitHub Copilot. The original quantitative analysis of the UVA data provides additional details on the programming tasks participants worked on during these trials [15]. In each block, participants saw up to eight tasks, where they could only view the next task after completing the current one. The ten participants at VSU were given identical tasks and AI treatments as ten of the participants at UVA. This study focuses only on the AI-assisted blocks and includes results using data from both sites. Trust data was not collected for solo trials as the IntelliSense autocomplete built into VS Code is deterministic and thus was not relevant to this study's research questions. As such, no comparison of trust in Copilot to a baseline of trust in IntelliSense was examined.

Before and after each AI-assisted block, participants completed a questionnaire containing a specific trust scale adapted from the work of Muir and Moray [45, 44]. For each trial, participants had 20 minutes to complete as many tasks as possible (up to eight). They were instructed to work for the best possible score, using the AI as much or little as needed. Bonus compensation was awarded to the top two scorers at each site to encourage adherence to this instruction. This was a key design choice for eliciting trust attitudes, as it created risk related to under-use

or over-use of AI assistance during the time-pressured scenario [33, 43]. Figure 4.2 shows the timeline of specific trust measurements and programming tasks. Note that because participants also programmed alone as part of a larger counterbalanced design, specific trust measurements 1B and 2B were either taken in close succession (AI trials were back-to-back) or separated by one or two trials without AI. Counterbalancing means systematically varying the order of task sets used for each trial and the choice of which order to assign AI or solo conditions.

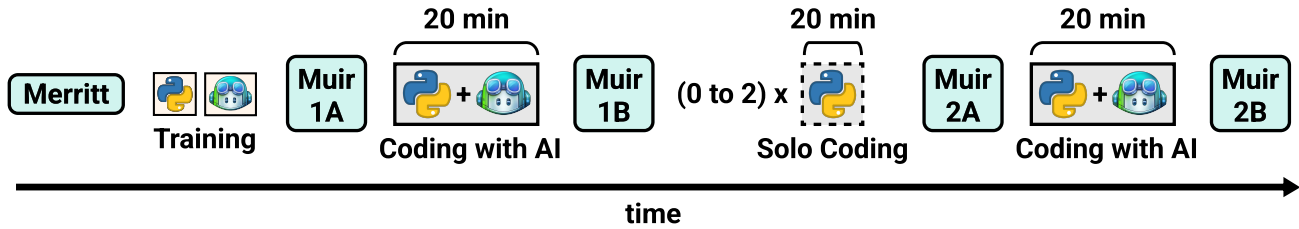


Figure 4.2: Timeline of trust scale administration amidst AI-assisted Python programming.

4.2.4 Measures of Trust, Compliance, and Performance

Trust

To measure trust in GitHub Copilot, we employed a specific trust scale adapted from the work of Muir [45, 44]. Muir and Moray concluded that perceptions of competence (see Equation (4.1)), followed by predictability, dependability, and faith (see Equation (4.2)) were key determinants of trust [45]. As shown in Table 4.1, we adapted both these four measurement items and the ground-truth overall trust item from [44] to our context. To avoid any ambiguity about the meaning of the word "trust" [4], this item was phrased as a perception of trustworthiness (reliability), which was clearly defined. The original scale was measured in millimeters on a paper instrument, so our digital version used a reasonable [41] zero to ten scale (0 = *'Not At All'* or *'Least Trustworthy'*, 10 = *'Completely'* or *'Most Trustworthy'*) with one point increments. Figure 4.3 shows an SEM path diagram for the five observed trust items; the measurement model contains four items as indicators, and the factor they indicate ought to correlate with the fifth (summary) item. Equation (4.3)

shows our application of Equations (4.1) and (4.2) that handles the multicollinearity discussed in [45].

$$(\text{Overall Trust Item}) = 14.1 + 0.8 \cdot \text{Competence} + \epsilon \quad (4.1)$$

for residual ϵ

$$(\text{Overall Trust Item}) = \beta_0 + \beta_1 \cdot \text{Predictability} + \beta_2 \cdot \text{Dependability} + \beta_3 \cdot \text{Faith} + \epsilon \quad (4.2)$$

for coefficients β and residual ϵ

Table 4.1: Subjective trust scale adapted from the Muir scale.

Item	Wording
Competence	How confident are you in Copilot to help you solve problems with code?
Predictability	How confident are you for Copilot to behave predictably?
Dependability	How confident are you in Copilot to help you avoid errors and bugs in your code?
Faith	Assuming you have access to it, how confident are you in Copilot to help you with future programming for work or school?
Overall Trust	Trustworthiness is the extent to which GitHub Copilot is useful enough and effective enough to rely on to help someone program. How trustworthy do you think GitHub Copilot is?

Note. The scale ranged from 0 to 10 with no decimals.

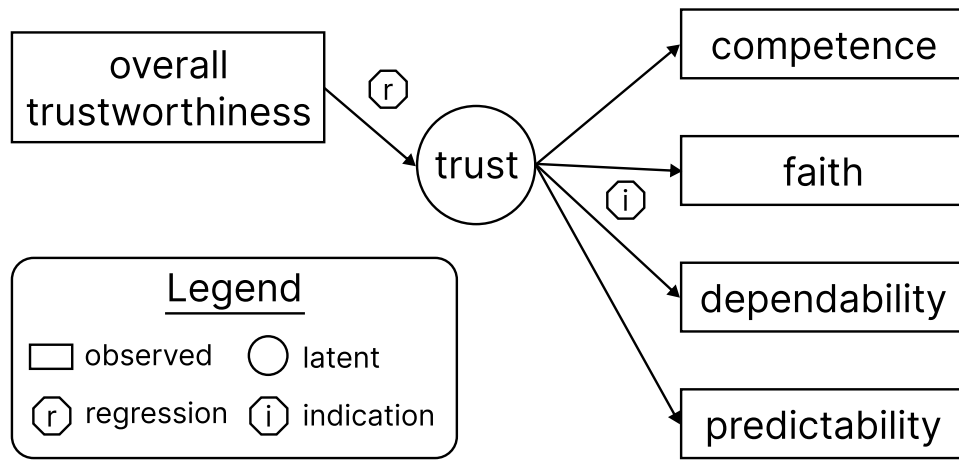


Figure 4.3: SEM path diagram for the adapted Muir scale.

Note. This Structural Equation Model (SEM) path diagram is composed of a four-indicator measurement model and an implied regression equation between an observed summary item (perceived trustworthiness) and the latent trust factor. We focused here on the measurement model only.

$$\begin{aligned}
 &(\text{Trust Factor}) = \gamma_1 \cdot (\text{Overall Trust Item}) + \zeta \\
 &\begin{bmatrix} \text{Competence} \\ \text{Predictability} \\ \text{Dependability} \\ \text{Faith} \end{bmatrix} = \begin{bmatrix} \lambda_C \\ \lambda_P \\ \lambda_D \\ \lambda_F \end{bmatrix} \cdot (\text{Trust Factor}) + \begin{bmatrix} \delta_C \\ \delta_P \\ \delta_D \\ \delta_F \end{bmatrix} \quad (4.3)
 \end{aligned}$$

for coefficient γ , residual ζ , loadings λ , and measurement errors δ

Compliance

Compliance to GitHub Copilot's advice was operationalized by counting the total number of times that participants used built-in compliance features during each trial. This included accepting inline suggestions, shown as greyed-out text, or using a button in the multiple suggestions panel to

accept one of its options. Notably, manual copy-and-pasting or typing of viewed suggestions was not counted. The newer version of Copilot in study two generated additional sources of compliances: (1) using buttons in the chat panel for copying or inserting provided code blocks and (2) accepting code edits from the inline chat widget. Once again, manual copy-pasting and typing were not counted. Only if the user clicked a built-in copy button was a copy tracked. It was important to count all sources of compliances from study two in case participants chose newer features in lieu of older ones.

Performance

Performance was measured using an ordinal score reflecting how many tasks a participant completed in 20 minutes (0-8 points). We were not sure about the measurement level of this data, as it is not possible to make tasks perfectly exchangeable in difficulty. Ultimately, we treated the score as a ratio-level count based on its nature and distribution. To ensure this assumption did not bias our conclusions, we also considered a binary transformation for all relevant regressions. That is, for the Conclusion Sensitivity Analysis, score was recoded as either *low* or *high* against the median of all trials with AI across all 27 participants (2 completed tasks of a possible 8). A *low* score was one at or below the median (0-2), while a *high* score was one above the median (3-8).

4.2.5 Data Analysis

All statistics were computed in R [20]. Unless otherwise stated, we used the standard statistical significance threshold of $\alpha = .05$.

Scale Validation and Factor Analysis

We assessed the internal consistency of the Muir scale using Cronbach's [9] α . Next, we performed single-factor Confirmatory Factor Analysis (CFA) to assess unidimensionality. Goodness-of-fit for CFA models was assessed using an accepted spectrum of fit measures (χ^2 p -value >

.05, CFI > .95, SRMR < .08, RMSEA < .06) [6, 22]. We compared two practically applicable CFA model structures using the Satorra-Bentler [61] scaled χ^2 difference test. First, we loaded each item onto the factor equally to represent the common application of collapsing the item ratings from a trust scale with an arithmetic mean. Next, we allowed the item loadings to be estimated freely. We used the better model to compute trust factor scores for further analysis, and assessed test-retest reliability with a paired t-test and the Pearson correlation coefficient [50].

Trust Score Computation

For computing trust scores, we applied the four-indicator CFA model using lavaan [58] with the `lavaan::lavPredict(type="ov")` function. This produced a score for each observed scale item response weighted by its contribution to the latent factor (determined by the CFA). We took row-wise means to produce latent factor scores distributed like the original data (not $\mu = 0, \sigma^2 = 1$).

Generalized Estimating Equations

We explored the interrelationships of trust, compliance, and performance using Generalized Estimating Equations (GEE). GEE extends Generalized Linear Model (GLM) regression to relax the assumptions of observational independence and family-consistent residual dispersion by accounting for clustering during estimation [30]. We also considered multi-level Bayesian regression, which aligns well to trust theory, but chose GEE as it is more robust and less sensitive to assumptions with as few clusters as our 27 [37, 51]. Compared to prior trust research that used pooled regression [28, 39], using GEE allowed us to explicitly account for dependence in our repeated-measures data.

Because we were not interested in using models for prediction, we were not concerned with the overall goodness-of-fit (i.e., R^2 or Information Criterion values). Rather, we were interested in the significance of each relationship and interpreting the coefficients. Therefore, we used coefficient Wald tests to assess statistical significance according to regression coefficients and standard errors. To control Type I error, we used `geesmv::GEE.var.mbn` [66] to compute Clus-

ter Robust Standard Errors (CRSEs) via the Morel-Bokossa-Neerchal [42] small-sample GEE correction [38, 37]. The coefficients themselves were estimated with `geepack::geeglm` [17] with identical settings as `geesmv`. The working correlation structure was set to "exchangeable" with the two observations per cluster used for RQ1 and RQ2 but to autoregressive ("ar1" for `geepack` and "AR-M" for `geesmv`) with the four trust observations per cluster used for RQ3.

The theoretical model in Figure 4.1 drove our choices for predictor and response measures, but we also included covariates for site and time. Trust scores were derived from the good-fit CFA models as described in Trust Score Computation and used as predictors for RQ1 and response measures for RQ3. Their symmetric distribution motivated a Gaussian family choice for RQ3. Compliances per trial were appropriately right skewed for a count process, motivating a Poisson family choice for RQ1. Performance score (at least ordinal; 0-8 points per trial) was right-skewed and treated as a Poisson process for RQ2. Given we used GEE and had more than seven discrete levels for both trust ratings and performance scores, relaxing assumptions about their measurement level was an ideal tradeoff over losing information and simplicity with an ordinal approach [55].

Conclusion Sensitivity Analysis

We considered the basic assumptions of GEE and the sensitivity of our conclusions (i.e., the direction and significance of our target predictors) to different reasonable assumptions. Though GEE is exempt from some regression assumptions, it is important to consider linearity, model specification, and influential observations. After choosing appropriate response distribution families, we confirmed general linearity by visual inspection. When specifying GLM formulas, we added covariates to account for time and research site. Next, we used `glmttoolbox::glmgee` [65] and Cook's [8] distance (stats) to check our conclusions for sensitivity to possible influential observations. Given we omitted the "Overall Trust" summary item of the Muir scale (see Scale Validation and Factor Analysis) and used CFA-derived trust factor scores in a non-SEM analysis, we also checked for conclusion sensitivity to using a simple mean of all five items instead. Finally, we checked sensitivity to treating ordinal performance data as ratio-level (see

Performance and Generalized Estimating Equations) using a binary transformation (i.e. at or below the median 0-2 points or above the median 3-8 points). This also called for a binomial family GEE when performance was the regression response. In each section of the Results to follow, we comment on the sensitivity of our results with respect to these three considerations: influential observations, trust aggregation, and performance measurement level. Otherwise, the data met the assumptions of GEE.

4.3 Results

4.3.1 Scale Validation and Factor Analysis

The Muir scale (see Table 4.1) produced Cronbach's $\alpha = .90$ with the "ground truth" trustworthiness item included and $\alpha = .87$ without it. This item was excluded from the measurement model on account of the covariance assumptions of CFA. The equal-weight four-indicator CFA fit poorly (Y-B $\chi^2(6) = 56.96$, $p < .001$, CFI_R $>.730$, RMSEA_R $= 0.305$, SRMR $= 0.381$). A freely estimated model of the four indicators was significantly better (S-B χ^2 Diff. $= 57.351$, $p <.001$), and the overall fit was good (Y-B $\chi^2(2) = 2.00$, $p = .37$, CFI_R $>.999$, RMSEA_R $= 0.002$, SRMR $= 0.022$). We used this model for computing trust scores at each timepoint. Participants had no new experiences with the AI between timepoints 1B and 2A (see Figure 4.2). Trust scores were test-retest reliable between these points according to the paired t -test ($t(26) = 0.520$, $p = .61$) and Pearson correlation ($r(25) = .93$, $p < .001$).

4.3.2 RQ1: Does specific trust in an AIDE predict subsequent compliance?

We first examined the relationship between the computed factor score for specific trust before a trial began (Figure 4.2, Trust 1A and 2A) and the number of times participants accepted text from Copilot using the built-in compliance features. With a Poisson family, pre-trial trust, along with covariates to control for trial number (1 v. 2) and research site (UVA v. VSU), was used as

a predictor of compliance count. Table 4.2 shows the results of the coefficient Wald tests based on the MBN CRSE correction. The exponentiation of the coefficients provides the multiplicative factor by which the models predict a single point of pre-trial specific trust (0 to 10) to scale the number of expected compliances during that trial. Pre-trial trust was not a statistically significant predictor of compliance, showing a slightly negative slope. The Conclusion Sensitivity Analysis revealed no sensitivity to trust aggregation or influential observations. Of note, the trust coefficient's direction shifted from -1% to +1% when two influential clusters were excluded. There is no evidence that pre-trial trust was related to the number of times participants used the built-in compliance features. This finding calls into question the link in Figure 4.1 between trust (a) and compliance (b). The estimated working correlation of $\hat{\alpha} = .83$ supports the suspected observational dependence structure. The statistically significant covariates suggest higher compliance at UVA than at VSU and higher compliance on the second trial than on the first.

Table 4.2: Exchangeable Poisson GEE estimates relating pre-trial trust to compliance count.

Variable	<i>B</i>	<i>e^B</i>	<i>CRSE</i>	95% CI	
				<i>LL</i>	<i>UL</i>
Constant	2.91***	$\hat{\mu} = 18$	0.32	2.29	3.54
Site = VSU	-0.87***	-58%	0.26	-1.37	-0.36
Trial Number	0.20**	+22%	0.07	0.06	0.33
Pre-Trial Trust (A)	-0.01	-1%	0.05	-0.10	0.08

Note. $k = 27$ clusters (participants) of balanced size $n_i = 2$ (trials per participant). We estimated the population-averaged relationship between the computed factor score for pre-trial trust and the number of compliances using Generalized Estimating Equations (GEE) with a Poisson link, exchangeable correlation structure, Morel-Bokossa-Neerchal (MBN) bias-corrected Cluster-Robust Standard Error (CRSE), and covariates for study site and time (i.e. trial number). The estimated working correlation was $\hat{\alpha} = .83$.

** $p < 0.01$; *** $p < 0.001$

4.3.3 RQ2: Does compliance predict performance when coding with an AIDE?

Pre-trial trust did not seem to relate to compliance, but we also wanted to see whether increased compliance led to any differences in performance. To test this, we again used Poisson GEE regression with an exchangeable correlation structure and the MBN correction. Table 4.2 shows the results of the coefficient Wald tests. There was a small but statistically significant positive relationship between compliance and performance ($p < .01$) corresponding to a roughly 2% improvement for each additional compliance ($\bar{x} \approx 19$, $SD \approx \tilde{x} \approx 14$). The significance of trial number suggests performance increased with additional experience regardless of the level of compliance. The Conclusion Sensitivity Analysis revealed little sensitivity. Excluding two influential clusters increased the compliance coefficient from 2% to 3%, reducing uncertainty slightly. Dichotomizing score data caused the trial number covariate to lose statistical significance, a possible indicator of entanglement between compliance and experience. Compliance was still significant, increasing the odds of scoring above the median by a roughly 9:8 ratio per compliance event. Figure 4.4 shows that participants were able to achieve high scores without correspondingly high levels of compliance, yet greater compliance clearly correlated with greater performance. The estimated working correlation of $\hat{\alpha} = .54$ indicates some correlation between each participant's own scoring rates on the two trials.

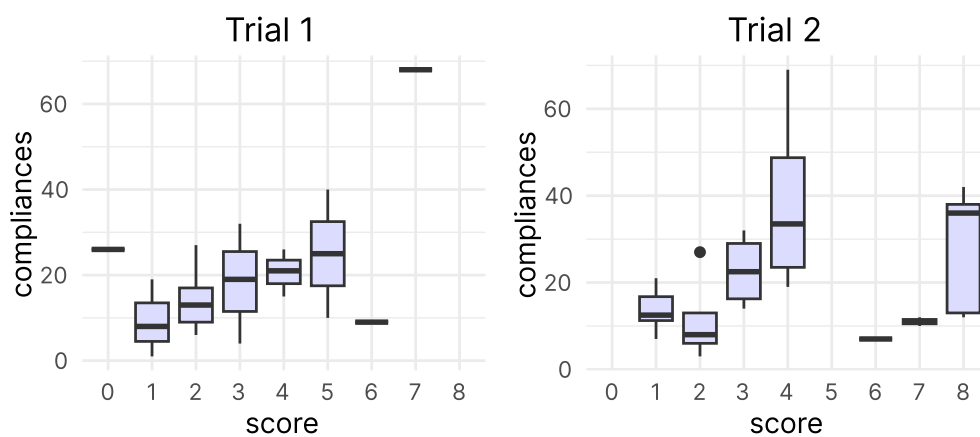


Figure 4.4: Boxplots of compliances at each possible score by trial.

Table 4.3: Exchangeable Poisson GEE estimates relating compliance count to performance score.

Variable	B	e^B	CRSE	95% CI	
				LL	UL
Constant	0.18	$\hat{\mu} = 1$	0.29	-0.39	0.74
Site = VSU	0.25	+28%	0.30	-0.35	0.84
Trial Number	0.34**	+41%	0.13	0.09	0.59
Compliances	0.02**	+2%	0.01	0.01	0.03

Note. $k = 27$ clusters (participants) of balanced size $n_i = 2$ (trials per participant). We estimated the population-averaged relationship between the number of compliance events and the performance score using Generalized Estimating Equations (GEE) with a Poisson link, exchangeable correlation structure, Morel-Bokossa-Neerchal (MBN) bias-corrected Cluster-Robust Standard Error (CRSE), and covariates for study site and time (i.e. trial number). The estimated working correlation was $\hat{\alpha} = .54$.

** $p < 0.01$

4.3.4 RQ3: Does programming performance predict subsequent specific trust in an AIDE?

To investigate the connection in Figure 4.1 from performance back to specific trust, we estimated a Gaussian GEE regression that included all four timepoints of specific trust scores. This allowed for the estimation of within-cluster correlation according to the autoregressive (AR1) structure supported by both trust research [56] and our Scale Validation and Factor Analysis. To differentiate the effects of a new experience from performance during that experience, we used a time factor covariate for whether the trust ratings were given pre- or post-trial. We then crossed this with score, keeping the product, but excluding the score-only predictor. Otherwise, the regression would nonsensically allow pre-trial trust to be predicted by a performance that had not yet happened (lookahead bias). Appropriately, the remaining product term goes to zero for pre-trial observations, with any systematic differences between pre- and post-trial ratings being captured by the time term. As shown in Figure 4.5, performance trended higher on trial two, so to avoid multicollinearity, no trial term (1 v. 2) was included. Table 4.4 shows the results of the coefficient Wald tests. There was a small but statistically significant positive relationship between perfor-

mance and subsequent trust, with each additional performance point (0-8) increasing trust by a fraction of a trust point (0-10). The significant covariates indicate generally higher trust ratings at VSU and decreases in trust following new AIDE experiences while controlling for performance. The interpretation is that it takes a score of $\frac{.46}{.17} \approx 3$ to maintain trust, with a lower or higher score changing trust accordingly. The Conclusion Sensitivity Analysis revealed minor sensitivity. Neither the removal of two influential clusters nor the inclusion of a trial number predictor had major impacts. Mean trust aggregation and performance dichotomization affected the significance of covariates but not the score term. The estimated working correlation of $\hat{\alpha} = .77$ supports the suspected autoregressive dependence structure.

Table 4.4: Autoregressive gaussian GEE estimates relating performance score to post-trial trust.

Variable	B	CRSE	95% CI	
			LL	UL
Constant	$\hat{\mu} = 6.32^{***}$	0.29	5.74	6.89
Site = VSU	1.17*	0.51	0.16	2.17
Post-Trial = True	-0.46*	0.22	-0.88	-0.03
(Post-Trial = T) x Score	0.17***	0.05	0.07	0.26

Note. $k = 27$ clusters (participants) of balanced size $n_i = 4$ (before and after each of two trials per participant). We estimated the population-averaged relationship between the performance score and the computed factor score for post-trial trust using Generalized Estimating Equations (GEE) with a Gaussian link, autoregressive (AR-1) correlation structure, Morel-Bokossa-Neerchal (MBN) bias-corrected Cluster-Robust Standard Error (CRSE), and covariates for study site and whether a given trust score followed a new programming trial. Pre-trial trust ratings were included, so the interaction term estimated the effect of performance while counteracting lookahead bias. The estimated working correlation was $\hat{\alpha} = .77$.

* $p < 0.05$; *** $p < 0.001$

4.4 Discussion

Overall, the results were inconclusive about a link from trust to AIDE compliance but did support relationships from compliance to performance and from performance to trust. Pre-interaction specific trust was not a significant predictor for AIDE compliance, but there was a positive rela-

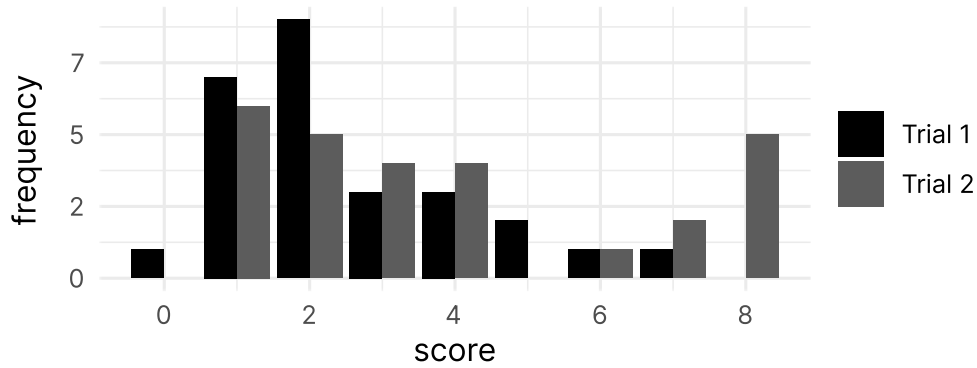


Figure 4.5: Histogram of scores by trial.

tionship between compliance and performance. Performance was also positively correlated to post-interaction trust.

4.4.1 RQ1: Does specific trust in an AIDE predict subsequent compliance?

We explored the relationship between pre-trial specific trust (1A and 2A in Figure 4.2) and the number of compliance events recorded during each trial. Here, we used trust as a predictor rather than a response in our GEE analysis. The negative sign of the estimated coefficient is opposite to the expectation from prior work [48], but the Wald test provided insufficient evidence to conclude whether the population-averaged rate of compliance events varied according to pre-trial specific trust. That is, the effect of trust on compliance was inconclusive. This outcome aligns to some recent findings that this effect, if present, is relatively small [48, 53]. The lack of alignment to earlier theory [29] and prior studies [10, 40, 68] should be interpreted as a motivation for additional research rather than complete evidence that compliance is unrelated to trust.

Our findings are perhaps best explained by the lack of risk associated with compliance in our study. Participants likely incurred very little marginal cost for each additional compliance. Prather et al. discussed this "no downside" [54] phenomenon in a qualitative study of novices using Copilot for the first time without time pressure. It is also consistent with the theory of automation bias [43]. In our study, as easily as participants accepted code, they could evaluate it

with pre-written tests and undo any changes they wished. Thus, a downside to compliance was the time cost of investing attention in Copilot's suggestions. Even a participant with low trust in Copilot could conceivably accept its suggestions as long as the potential upside was worth the time cost. In a scenario like that reported by Ross et al. [57] where the performance of code could not be assessed objectively, users might comply less due to the momentary uncertainty about the marginal value of accepting a suggestion. Fundamentally, trust in automation is characterized by uncertainty about its true reliability [29], such that feedback mechanisms aligned with the user's goals (e.g. the very same unit tests used to measure the performance they are striving toward) can downplay the role of trust in determining compliance behaviors. In [28], participants' strong tendency for self-reliance was only overcome when the operator's trust in automation was very high compared to their self-confidence. They were not incentivized to risk the downsides of the automated machinery controllers performing poorly unless they did not think they could manage them manually. By comparison, our participants were incentivized to comply with the AIDE because, aside from time cost, they only stood to gain from its help.

The inconclusive relationship between trust and compliance might also be explained by the limitations of our compliance measure. While the kind of compliance we measured might not relate to trust, some other kind might. As described in the Method section, we counted a compliance event whenever a user used a built-in compliance feature to accept code from Copilot. This included tab completions and clickable code insertion and editing tools. However, if participants complied with Copilot's suggestions by manually typing them out or copy-and-pasting suggested code, our data collection system did not record this. Whereas "slow accepting" [54] suggestions by typing them out character-by-character, which was not tracked, was an investment of time perhaps risky enough to be done on the basis of trust, the counted compliances—those triggered by pressing the tab key or clicking a button—were presumably faster, less risky, and thus maybe less related to trust. Finally, compliance is easily reversible by discarding accepted code. Sabouri et al. [59] observed 10 developers using AI assistance and found they kept only about half of the code they first accepted. Future work should investigate whether discounting compliances according to reversals affects the relationship to initial trust.

4.4.2 RQ2: Does compliance predict performance when coding with an AIDE?

We next explored the relationship between compliance and performance and found a significant positive relationship between compliance and performance scoring rate, with each additional compliance event (range 1 to 69) corresponding to about a 2% performance increase. Prior work suggested that this could be the case if the AIDE was sufficiently reliable to align with how much participants chose to use it [26, 29, 35, 47]. On such simple tasks, Copilot should have been highly reliable [14]. As discussed above, for reliability in this scenario to be sufficient, suggestions needed to contain enough useful insights to help participants produce correct code without costing too much time. Even as participants were expected to develop their own strategies during the experiment, those who used the AIDE more seemed to have benefited from doing so. This finding emphasizes the risk of automation disuse (under-reliance), which has often been eclipsed in existing literature by the risk of misuse (over-reliance) [25, 46, 54, 57, 64]. When someone underuses AI-generated content, the performance of the human-AI team can suffer as a result, regardless of the AI's true reliability for the task at hand.

4.4.3 RQ3: Does programming performance predict subsequent specific trust in an AIDE?

Prior work suggested that if participants had a sense of how they were performing with the AIDE, that their trust would follow suit [12, 28, 29, 40, 68, 69], and our results confirmed this expectation. Here, we used trust as a response rather than a predictor in our GEE analysis. We found a significant positive relationship between performance score and post-trial trust ratings while accounting for the autoregressive nature of trust. The practical significance of the observed effect is small at less than 1% of the trust scale's range per additional point scored of eight. We know from our previous quantitative analysis that participants from UVA performed significantly better on their second trial with the AIDE compared to the first [15]. While the GEE approach accounted for the correlation of participants' trust ratings over time, a possible explanation for the

relatively small observed effect of performance is the idea that trust has "inertia" [19, 28]. With trust "inertia," participants who had trust-building experiences in trial one might not have fully adjusted their trust until later. Especially if participants felt their trust was betrayed in some way in trial one, either the accumulation of positive experiences from both trials or an inertia of positive experiences from trial one over time may have been needed before their trust would increase.

4.4.4 Implications

Understanding trust is useful for an engineer if: (a) it affects a metric of importance to their organization and (b) it can be changed through a feasible intervention. We focused on the former by assessing how a novice's trust in an AIDE affects their compliance and performance on programming tasks. With the low-risk conditions in our study and the limitations of our compliance measure, we could not conclude whether participants' pre-trial specific trust had any meaningful impact on compliance. Therefore, when considering novices using AIDEs in performance-oriented task environments, our results did not suggest whether measures to manipulate trust will be useful in changing compliance behavior. However, compliance was positively associated with performance, so if maximal performance is the goal, training should focus on discouraging disuse in addition to discouraging misuse. Trust is a multifaceted attitude that may relate to other metrics of interest, such as workload and emotion, that were not studied here [10, 39]. Our findings indicate that novices trusted the AIDE more when it helped them perform well. To improve novices' trust outside of training materials, engineers can also focus on the ongoing effort toward making AIDEs themselves more reliable and trustworthy [4, 18].

4.4.5 Limitations

The primary limitations of our study were how we measured compliance and the design choice to give participants a performance feedback tool (unit tests). As described in the Method, we were only able to measure compliance events when participants used built-in compliance features, so there is a possibility that trust was related to copy-paste or typing initiated compliances in a way

we were not able to observe. Furthermore, participants' easy access to performance feedback limited the all-important influence of uncertainty on specific trust [16, 29]. While sample size and generalizability are always challenges, we were able to support some theories about trust derived from prior human factors research. The higher trust ratings and lower compliance rates at VSU compared to UVA emphasize the need for caution when generalizing over time and into varied contexts. Because we did not measure learning, we cannot make claims regarding the role of trust in effective learning, but we encourage future work in this area. Finally, given practitioners may intend to use this work to inform trust-targeted interventions, we should emphasize that our findings rely only on natural variations in trust that happen during AIDE interactions. By contrast, when trust is intentionally manipulated, it may vary to a greater extent, uncovering an otherwise hidden relationship with compliance [3, 63].

4.5 Conclusion

Conventional wisdom advises placing appropriate levels of trust in automation to avoid misuse or disuse. There is a contextually varied gap between what automation can do and whether humans think they can rely on it. In theory, this determines whether they do rely on it, creating a feedback loop [29]. We assessed this theory in the context of novice programmers using GitHub Copilot under time pressure. We measured specific trust in Copilot, compliance to its suggestions, and programming performance to test the validity of theory in our context. We failed to find a conclusive link from compliance to performance. However, compliance and performance were positively associated, and higher performance led to increased trust. These findings indicate that trust is related to performance and compliance but that manipulating trust to improve the performance of hybrid intelligence teams is a questionable approach that warrants further study. Practitioners should seek out more direct interventions for their metrics of interest and continue to make AIDEs more trustworthy programming companions.

4.6 Key Points

- 27 novice programmers at two research sites programmed with AI (GitHub Copilot) under time pressure for two 20-minute trials
- Trust in GitHub Copilot was measured four times with an adaptation of the Muir scale, performance was measured based on number of tasks completed per trial, and compliance was measured when Copilot's built-in compliance features were used
- Pre-trial trust failed to predict measures of compliance
- Performance was positively associated with AI compliance and post-trial trust

Acknowledgments

This work was supported by the University of Virginia Distinguished Fellowship, NSF National Research Traineeship for Cyber-Physical Systems (1829004), Graduate Research Fellowship Program for Computer and Information Science and Engineering (1842490), and the Virginia Commonwealth Cyber Initiative Central Virginia Node (GR102900).

References

- [1] Icek. Ajzen and Martin. Fishbein. 1980. *Understanding attitudes and predicting social behavior : illustration of applied social research*. Prentice-Hall, Englewood Cliffs, N.J. ISBN: 0-13-936435-8.
- [2] Matin Amoozadeh, David Daniels, Daye Nam, Aayush Kumar, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Trust in generative AI among students: an exploratory study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024: The 55th ACM Technical Symposium on Computer Science Education. ACM, Portland OR USA, (Mar. 7, 2024), 67–73. ISBN: 979-8-4007-0423-9. doi:10.1145/3626252.3630842.

- [3] David P. Biros, Gregory Fields, and Gregg Gunsch. 2003. The effect of external safeguards on human-information system trust in an information warfare environment. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. IEEE, 10–pp. Retrieved Oct. 25, 2024 from <https://ieeexplore.ieee.org/abstract/document/1173894/>.
- [4] Matthew L. Bolton. 2022. Trust is not a virtue: why we should not trust trust. *Ergonomics in Design*, 10648046221130171. Publisher: SAGE Publications Sage CA: Los Angeles, CA.
- [5] Adam Brown, Sarah D'Angelo, Ambar Murillo, Ciera Jaspan, and Collin Green. 2024. Identifying the factors that influence trust in AI code completion. In *Proceedings of the 1st ACM International Conference on AI-Powered Software (Alware 2024)*. Association for Computing Machinery, New York, NY, USA, (July 10, 2024), 1–9. ISBN: 979-8-4007-0685-1. doi:10.1145/3664646.3664757.
- [6] Barbara M. Byrne and Barbara M. Byrne. 2006. *Structural Equation Modeling With EQS: Basic Concepts, Applications, and Programming, Second Edition*. (2nd ed.). Routledge, New York, (Feb. 16, 2006). 454 pp. ISBN: 978-0-203-72653-2. doi:10.4324/9780203726532.
- [7] Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- [8] R. Dennis Cook. 1977. Detection of influential observation in linear regression. *Technometrics*, 19, 1, (Feb. 1977), 15–18. doi:10.1080/00401706.1977.10489493.
- [9] Lee J. Cronbach. 1951. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16, 3, (Sept. 1951), 297–334. doi:10.1007/BF02310555.
- [10] Mark Daly. 2002. *Task Load and Automation Use in an Uncertain Environment*. Ph.D. Dissertation. (Mar. 1, 2002). <https://scholar.afit.edu/etd/4394>.
- [11] Stephen R. Dixon, Christopher D. Wickens, and Jason S. McCarley. 2007. On the independence of compliance and reliance: are automation false alarms worse than misses? *Human Factors*, 49, 4, (Aug. 1, 2007), 564–572. doi:10.1518/001872007X215656.
- [12] Mary T. Dzindolet, Scott A. Peterson, Regina A. Pomranky, Linda G. Pierce, and Hall P. Beck. 2003. The role of trust in automation reliance. *International journal of human-computer studies*, 58, 6, 697–718. Retrieved Oct. 24, 2024 from <https://www.sciencedirect.com/science/article/pii/S1071581903000387>.
- [13] Guangrui Fan, Dandan Liu, Rui Zhang, and Lihu Pan. 2025. The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, 12, 1, (Mar. 4, 2025), 16. doi:10.1186/s40594-025-00537-3.

- [14] 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 (ACE '23)*. Association for Computing Machinery, New York, NY, USA, (Jan. 30, 2023), 97–104. ISBN: 978-1-4503-9941-8. doi:10.1145/3576123.3576134.
- [15] Nicholas Gardella, Raymond Pettit, and Sara L. Riggs. 2024. Performance, workload, emotion, and self-efficacy of novice programmers using AI code generation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. ITICSE 2024: Innovation and Technology in Computer Science Education*. ACM, Milan Italy, (July 3, 2024), 290–296. ISBN: 979-8-4007-0600-4. doi:10.1145/3649217.3653615.
- [16] Kevin Anthony Hoff and Masooda Bashir. 2015. Trust in automation: integrating empirical evidence on factors that influence trust. *Human Factors*, 57, 3, (May 1, 2015), 407–434. doi:10.1177/0018720814547570.
- [17] Søren Højsgaard, Ulrich Halekoh, and Jun Yan. 2006. The r package geepack for generalized estimating equations. *Journal of statistical software*, 15, 1–11. Retrieved Jan. 29, 2026 from <https://www.jstatsoft.org/article/view/v015i02>.
- [18] Dwight Horne. 2023. PwnPilot: reflections on trusting trust in the age of large language models and AI code assistants. In *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*. IEEE, 2457–2464. Retrieved Oct. 14, 2024 from <https://ieeexplore.ieee.org/abstract/document/10487488/>.
- [19] Wan-Lin Hu, Kumar Akash, Tahira Reid, and Neera Jain. 2019. Computational modeling of the dynamics of human trust during human–machine interactions. *IEEE Transactions on Human-Machine Systems*, 49, 6, (Dec. 2019), 485–497. doi:10.1109/THMS.2018.2874188.
- [20] Ross Ihaka and Robert Gentleman. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5, 3, (Sept. 1996), 299–314. doi:10.1080/10618600.1996.10474713.
- [21] [n. d.] Investigating and designing for trust in AI-powered code generation tools | proceedings of the 2024 ACM conference on fairness, accountability, and transparency. ACM Other conferences. Retrieved Jan. 30, 2026 from <https://dl.acm.org/doi/10.1145/3630106.3658984>.
- [22] Dennis L. Jackson, J. Arthur Gillaspay Jr, and Rebecca Purc-Stephenson. 2009. Reporting practices in confirmatory factor analysis: an overview and some recommendations. *Psychological methods*, 14, 1, 6. Retrieved Nov. 6, 2024 from <https://psycnet.apa.org/fulltext/2009-02702-005.html>.

- [23] 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*, 1–15.
- [24] 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, (Apr. 19, 2023), 1–23. ISBN: 978-1-4503-9421-5. doi:10.1145/3544548.3580919.
- [25] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–12. ISBN: 979-8-4007-1653-9. doi:10.1145/3631802.3631806.
- [26] Artur Klingbeil, Cassandra Grützner, and Philipp Schreck. 2024. Trust and reliance on AI—an experimental study on the extent and costs of overreliance on AI. *Computers in Human Behavior*, 160, 108352. Retrieved Oct. 16, 2024 from <https://www.sciencedirect.com/science/article/pii/S0747563224002206>.
- [27] Roderick M. Kramer. 1999. TRUST AND DISTRUST IN ORGANIZATIONS: emerging perspectives, enduring questions. *Annual Review of Psychology*, 50, 1, (Feb. 1999), 569–598. doi:10.1146/annurev.psych.50.1.569.
- [28] John Lee and Neville Moray. 1992. Trust, control strategies and allocation of function in human-machine systems. *Ergonomics*, 35, 10, (Oct. 1992), 1243–1270. doi:10.1080/00140139208967392.
- [29] John D. Lee and Katrina A. See. 2004. Trust in automation: designing for appropriate reliance. *Human factors*, 46, 1, 50–80. Publisher: SAGE Publications Sage UK: London, England.
- [30] Kung-Yee Liang and Scott L. Zeger. 1986. Longitudinal data analysis using generalized linear models. *Biometrika*, 73, 1, 13–22. doi:10.2307/2336267.
- [31] Jack Lindsey et al. 2025. On the biology of a large language model. *Transformer Circuits Thread*. <https://transformer-circuits.pub/2025/attribution-graphs/biology.html>.
- [32] Stephen Marsh and Mark R. Dibben. 2003. The role of trust in information science and technology. *Annual Review of Information Science and Technology (ARIST)*, 37, 465–98. Retrieved Apr. 18, 2025 from <https://www.learntechlib.org/p/96318/>.
- [33] Roger C. Mayer, James H. Davis, and F. David Schoorman. 1995. An integrative model of organizational trust. *The Academy of Management Review*, 20, 3, 709–734. doi:10.2307/258792.

- [34] Michael McCracken et al. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working group reports from ITiCSE on Innovation and technology in computer science education* (ITiCSE-WGR '01). Association for Computing Machinery, New York, NY, USA, (Dec. 1, 2001), 125–180. ISBN: 978-1-4503-7359-3. doi:10.1145/572133.572137.
- [35] John M. McGuirl and Nadine B. Sarter. 2006. Supporting trust calibration and the effective use of decision aids by presenting dynamic system confidence information. *Human Factors*, 48, 4, (Dec. 1, 2006), 656–665. doi:10.1518/001872006779166334.
- [36] D. McKnight and Norman Chervany. 2000. What is trust? a conceptual analysis and an interdisciplinary model. *AMCIS 2000 Proceedings*, (Jan. 1, 2000). <https://aisel.aisnet.org/amcis2000/382>.
- [37] Daniel McNeish and Laura M. Stapleton. 2016. Modeling clustered data with very few clusters. *Multivariate Behavioral Research*, 51, 4, (July 3, 2016), 495–518. doi:10.1080/00273171.2016.1167008.
- [38] Daniel M. McNeish. 2014. Modeling sparsely clustered data: design-based, model-based, and single-level methods. *Psychological Methods*, 19, 4, 552–563. doi:10.1037/met0000024.
- [39] Stephanie M. Merritt. 2011. Affective processes in human–automation interactions. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 53, 4, (Aug. 2011), 356–370. doi:10.1177/0018720811411912.
- [40] Stephanie M. Merritt and Daniel R. Ilgen. 2008. Not all trust is created equal: dispositional and history-based trust in human-automation interactions. *Human factors*, 50, 2, 194–210. Number: 2 Publisher: SAGE Publications Sage CA: Los Angeles, CA.
- [41] George A. Miller. 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63, 2, (Mar. 1956), 81–97. Number: 2. doi:10.1037/h0043158.
- [42] J.g. Morel, M.c. Bokossa, and N.k. Neerchal. 2003. Small sample correction for the variance of GEE estimators. *Biometrical Journal*, 45, 4, 395–409. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/bimj.200390021>. doi:10.1002/bimj.200390021.
- [43] Kathleen L. Mosier and Linda J. Skitka. 1999. Automation use and automation bias. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 43, 3, (Sept. 1999), 344–348. doi:10.1177/154193129904300346.
- [44] Bonnie M. Muir. 1989. *Operators' trust in and use of automatic controllers in a supervisory process control task*. Ph.D. Dissertation. University of Toronto (Canada), Canada – Ontario, CA. ISBN: 978-0-315-51014-2. Retrieved June 29, 2023 from <https://www.proquest.com/pqdtglobal/docview/303794539/citation/EA1C186F755147DBPQ/1>.

- [45] Bonnie M. Muir and Neville Moray. 1996. Trust in automation. part II. experimental studies of trust and human intervention in a process control simulation. *Ergonomics*, 39, 3, (Mar. 1996), 429–460. doi:10.1080/00140139608964474.
- [46] Raja Parasuraman and Victor Riley. 1997. Humans and automation: use, misuse, disuse, abuse. *Human Factors*, 39, 2, (June 1, 1997), 230–253. doi:10.1518/001872097778543886.
- [47] Samir Passi and Mihaela Vorvoreanu. 2022. Overreliance on AI literature review. *Microsoft Research*. Retrieved Oct. 14, 2024 from <https://www.microsoft.com/en-us/research/uploads/prod/2022/06/Aether-Overreliance-on-AI-Review-Final-6.21.22.pdf>.
- [48] Colleen E. Patton and Christopher D. Wickens. 2024. The relationship of trust and dependence. *Ergonomics*, 67, 11, (Nov. 2024), 1535–1552. doi:10.1080/00140139.2024.2342436.
- [49] 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.
- [50] Karl Pearson and Francis Galton. 1997. VII. note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58, 347, (Jan. 1997), 240–242. doi:10.1098/rsp1.1895.0041.
- [51] Claudia Pedroza and Van Thi Thanh Truong. 2017. Estimating relative risks in multicenter studies with a small number of centers — which methods to use? a simulation study. *Trials*, 18, 1, (Dec. 2017), 512. doi:10.1186/s13063-017-2248-1.
- [52] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, (Nov. 21, 2023), 2785–2799. ISBN: 979-8-4007-0050-7. doi:10.1145/3576915.3623157.
- [53] Rebecca L. Pharmer, Christopher D. Wickens, and Benjamin A. Clegg. 2025. Transparent systems, opaque results: a study on automation compliance and task performance. *Cognitive Research: Principles and Implications*, 10, 1, (Feb. 21, 2025), 8. doi:10.1186/s41235-025-00619-4.
- [54] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. “it’s weird that it knows what i want”: usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 29, 2024), 1–31. doi:10.1145/3617367.

- [55] Mijke Rhemtulla, Patricia É Brosseau-Liard, and Victoria Savalei. 2012. When can categorical variables be treated as continuous? a comparison of robust continuous and categorical SEM estimation methods under suboptimal conditions. *Psychological methods*, 17, 3, 354. Retrieved Jan. 28, 2026 from <https://psycnet.apa.org/record/2012-18631-001>.
- [56] Lucero Rodriguez Rodriguez, Carlos E. Bustamante Orellana, Erin K. Chiou, Lixiao Huang, Nancy Cooke, and Yun Kang. 2023. A review of mathematical models of human trust in automation. *Frontiers in Neuroergonomics*, 4, 1171403. Retrieved Jan. 30, 2026 from <https://www.frontiersin.org/journals/neuroergonomics/articles/10.3389/fnrgo.2023.1171403/full>.
- [57] 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. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. IUI '23: 28th International Conference on Intelligent User Interfaces. ACM, Sydney NSW Australia, (Mar. 27, 2023), 491–514. ISBN: 979-8-4007-0106-1. doi:10.1145/3581641.3584037.
- [58] Yves Rosseel. 2012. Lavaan: an r package for structural equation modeling. *Journal of statistical software*, 48, 1–36. Retrieved Nov. 18, 2024 from <https://www.jstatsoft.org/article/view/v048i02>.
- [59] Sadra Sabouri, Philipp Eibl, Xinyi Zhou, Morteza Ziyadi, Nenad Medvidovic, Lars Lindemann, and Souti Chattopadhyay. 2025. Trust dynamics in AI-assisted development: definitions, factors, and implications. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE). ISSN: 1558-1225. (Apr. 2025), 1678–1690. doi:10.1109/ICSE55347.2025.00199.
- [60] Caitlin Sadowski and Thomas Zimmermann. 2019. *Rethinking productivity in software engineering*. Springer Nature. Retrieved Sept. 9, 2024 from <https://library.oapen.org/handle/20.500.12657/22839>.
- [61] Albert Satorra and Peter M. Bentler. 2001. A scaled difference chi-square test statistic for moment structure analysis. *Psychometrika*, 66, 4, 507–514. Retrieved Feb. 12, 2025 from https://idp.springer.com/authorize/casa?redirect_uri=https://link.springer.com/article/10.1007/bf02296192&casa_token=u9u5q4_uXUkAAAAA:EVVL_ANPhHHleRCeBtfDARzrPJosFphuy3vIahZxBWiYYthr3yTPPkYHRHIFyio_0kKLcxMKGL8q5ulmyw.
- [62] Anshul Shah, Anya Chernova, Elena Tomson, Leo Porter, William G. Griswold, and Adalbert Gerald Soosai Raj. 2025. Students’ use of GitHub copilot for working with large code bases. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, (Feb. 18, 2025), 1050–1056. ISBN: 979-8-4007-0531-1. doi:10.1145/3641554.3701800.

- [63] Randall D. Spain. 2009. The effects of automation expertise, system confidence, and image quality on trust, compliance, and performance. Retrieved Oct. 25, 2024 from https://digitalcommons.odu.edu/psychology_etds/120/.
- [64] 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 (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, (Apr. 28, 2022), 1–7. ISBN: 978-1-4503-9156-6. doi:10.1145/3491101.3519665.
- [65] Luis Hernando Vanegas, Luz Marina Rondon, and Gilberto Alvarenga Paula. 2023. Generalized estimating equations using the new r package glmttoolbox. *The R Journal*, 15, 2, 105–133.
- [66] [SW] Ming Wang, Geesmv: Modified Variance Estimators for Generalized Estimating Equations version 1.3, 2015. url: <https://cran.r-project.org/web/packages/geesmv/index.html>.
- [67] Justin D. Weisz et al. 2021. Perfection not required? human-AI partnerships in code translation. In MAG ID: 3154248444. (Apr. 14, 2021), 402–412. doi:10.1145/3397481.3450656.
- [68] Jacob M. Wetzel. 2005. *Driver trust, annoyance, and compliance for an automated calendar system*. Ph.D. Dissertation. Central Michigan University, United States – Michigan. 218 pp. isbn: 978-0-542-38232-1. Retrieved Oct. 25, 2024 from <https://www.proquest.com/docview/305363652/abstract/C9A2EE5257934DCBPQ/1>.
- [69] X. Jessie Yang, Christopher Schemanske, and Christine Searle. 2023. Toward quantifying trust dynamics: how people adjust their trust after moment-to-moment interaction with automation. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 65, 5, (Aug. 2023), 862–878. doi:10.1177/00187208211034716.

Chapter 5

HBCU Student Perspectives on Identity, Persistence, and Code-Generating AI in CS Education: A Case Study

Chapter Preamble

This chapter is the first part of Aim 2, situating novices' experiences programming with Copilot against the societal context of racial exclusion in computing education. While Aim 1 focused on evaluating and explaining interactions, this chapter begins assessing what the nature of interactions says about the implications of Copilot for computing students. Though the findings here did not seem entirely specific to Black HBCU students, this deliberate effort to include underrepresented voices contributes to the overall mission to promote educational equity with the policy recommendations offered in Chapter 8. Furthermore, several educational interventions beyond AI adoption are considered, as is the overall social context into which this research fits.

Authorship, Reuse, and Licensing Statement With the explicit consent of coauthors Isabella Graßl , Joseph Shelton , and Sara L. Riggs , this chapter directly reuses contents from [31], a conference proceedings paper led by Nicholas Gardella . The license for this chapter is CC-BY 4.0. ACM allows complete reuse of published papers in doctoral dissertations per the following section of the *ACM Author Rights* policy, retrieved on March 16, 2026: "Authors can include partial or complete papers of their own in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included." Additionally, this chapter explores data from a study that also contributed to the datasets used in Chapters 4 and 6; Chapters 4 to 6 all analyze different research questions, and none of the data reported here is reported for any other study.

Abstract

Code-generating Artificial Intelligence (AI) is disrupting traditional computer science (CS) education with unclear implications for underrepresented students. While these tools could democratize CS education, they may also widen existing privilege gaps. Little research examines how Black students at Historically Black Colleges and Universities (HBCUs) view code-generating AI in relation to their identity and persistence in CS. We present a case study of Black HBCU students' perspectives on identity and persistence factors in relation to using code-generating AI, identifying implications for educational equity. We recruited 10 Black CS undergraduates from an HBCU to solve basic Python programming tasks with and without GitHub Copilot, using a pre-study questionnaire and a semi-structured exit interview to gather their perspectives. Participants emphasized three key identity aspects: expressing their humanity, striving for social impact, and navigating social categories. Success factors included gratification, social support, self-teaching, and financial aid, while barriers were deficits compared to others, and social barriers. Students expected AI-tools to help overcome deficits and social barriers and offer moral support, but anticipated drawbacks such as exposing deficits, disrupting identities, hindering learning, and promoting antisocial tendencies. Our findings underscore that personhood and solidarity are key to CS persistence for Black HBCU students. While AI can augment pedagogy and address deficits, it cannot replace human community. We recommend conservative deployment of AI-tools in CS education and caution against harms to learning, identity, and social life from constant use. Participants' undervaluing of premium AI poses a risk for technological privilege gaps to widen further.

5.1 Introduction

Black students are underrepresented in both CS education and computing professions despite decades of mitigating efforts [1, 28]. Being as CS education research recruits current students, early work about the impacts of code-generating AI on students does not adequately represent

the perspectives of those from underrepresented backgrounds. Research has explained many factors that contribute to students' interest and retention in CS [2, 6, 20, 37]. Prior work has also revealed that many barriers disproportionately affect minoritized students [3, 14, 58, 59, 60]. However, new generative AI products may change how students navigate CS education.

Recent advances in AI code generators have raised concerns about whether this progress stands to help or hinder racial and gender equity in CS [42, 52, 54]. While prior research has illustrated how identity and persistence factors relate to diversity in computing, there is a major gap regarding what impact new AI tools will have on this nexus [55]. HBCUs, where Black post-secondary students in the U.S. can access culturally affirming education, are one useful site to study these factors. A first step for increasing Black inclusion in CS amidst the adoption of AI code generators is gathering and adapting to Black students' perspectives.

In this vein, our emancipatory-style [64] qualitative case study investigates identity, persistence factors, and the impact of AI thereon. Ten novice programmers were recruited from second-level Introduction to Programming (CS2) classes at a public HBCU in the U.S. They programmed in Python under time pressure with and without assistance from the GitHub Copilot Artificial Intelligence-driven Development Environment (AIDE) and participated in a semi-structured interview informed by the work of Rankin et al. [59]. The study focused on the following research questions:

- **RQ1:** How do HBCU CS students view their personal and computing identities?
- **RQ2:** What factors do HBCU CS students perceive to be helpful and harmful to persistence in computing?
- **RQ3:** How do HBCU CS students think about AIDEs in relation to identity and persistence in computing?

Our case study reveals that Black HBCU students prioritize communal goals and humanism within their identities. The persistence of Black HBCU students can be helped by self-motivated and gratifying educational experiences that still prioritize connections with others and the societal purpose of computing work. Participants reveal how code-generating AI helps them face their

deficits, which is simultaneously helpful and demoralizing, while socially, it both isolates and protects them.

These findings suggest that effective CS education for underrepresented students must balance technological innovation with human connection. We recommend deploying AI-tools strategically, using them to help students overcome knowledge deficits and social barriers while preserving other opportunities for identity development and social life.

5.2 Positionality Statement

The authors share the position that diversity in computer science is inherently valuable for increasing human innovative potential and promoting equity in society through technology. All but one of the authors identify within groups historically underrepresented in CS, and we all strive to use our relative privilege to amplify the voices of Black students and resist *color blind* pedagogy.

5.3 Theoretical Framework

5.3.1 Myth of Meritocracy

Our first theoretical basis is that success in computing is not based on merit alone but on the fortune of being born into a dominant social group. One explanation of this concept is Bourdieu's theory of reproduction, which explains that the education system is a tool for dominant groups to perpetuate their overrepresentation by gatekeeping educational credentials according to their own culture [8]. Since academia in the U.S. is dominated by White educators, what is considered worth learning is inherently biased away from Black culture. This challenges the perspective that an individual's interest and investment in their own education is the key determinant of their success [18, 69]. In contrast to a "colorblind" meritocracy, an inclusive computing field is one that acknowledges Black culture and values. For example, the Culturally Engaging Campus Environments (CECE) model argues that students will be more likely to feel they belong and to persist to

graduation when their learning environment represents their culture and helps them to engage with it and others in a collectivist manner [50]. CECE's emphasis on collectivism, humanism, and community service captures how students need a socially rich educational environment to succeed.

5.3.2 Goal Incongruity

Our second theoretical basis concerns how diversity in CS is harmed by its perception as anti-social and incongruous with communal goals. According to Holland's vocational theory, people seek out work environments that match their personal identities [36], yet only by entering computing spaces can people adjust their personal identities to align with the typical "investigative" [36] personality associated with CS [62]. If one has a "social" [36] personality that emphasizes helping and communing with others, CS may not seem to be a good fit for them. This further justifies how a CECE approach can help overcome cycles of cultural reproduction by embracing social aspects of education. Diekman's theory of Goal-Congruity traces the lack of diversity in STEM, specifically, to its poor expression of communal goals—people's desires to collaborate with and help others. By this theory, people who are underrepresented in STEM fail to see or experience an alignment of STEM professions with communal goals [22]. Empirical work focused on computing has confirmed that students who value communal goals have a greater sense of belonging when they experience communal goal congruity; furthermore, Black students and women have been found to value communal goals more than White students and men [38]. Overall, these theories indicate that increasing Black representation in CS can be helped by reshaping computing to emphasize social connectedness and service for others.

5.4 Related Work

5.4.1 Identity

White, Asian, and male identities are overrepresented in the U.S. computing field [1, 62]. Into the 2020s, it remains undeniable that bachelor's degrees in computing are awarded disproportionately relative to the makeup of the U.S. population [17, 11, 12]. Despite comprising over 50% of the population [11], women received just 23% of 2022 computing bachelor's degrees [17]. Of the largest racial categories in the U.S. (White, Hispanic/Latinx, Black, and Asian) Asian graduates were over-represented by nearly 200%. Consequently Hispanic/Latinx, Black, and White graduates were underrepresented by 39%, 35%, and 22% respectively (relative to their proportion within the population). Furthermore, graduates with intersectional [16] identities (e.g. Black and gender-female) were underrepresented by margins in excess of 60% [17, 11, 12].

Fostering students' personal identities within computing is critical for their success in the field [43, 62]. A 2024 literature review identified four aspects of a computing identity: performance, responsibility, motivation, and recognition [33]. The key takeaway is that a computing identity arises when one feels a sense of belonging in CS and a calling to translate computing interests and skills into societal impacts. Identity more broadly considers one's sense of self. This can, but does not have to, include social categories like race, gender, or national origin. While seemingly individual, social interactions also help to shape computing identity through recognition of one's apparent fit to the computing culture around them [51, 59, 62, 71]. In line with the theories discussed in Section 5.3.1, the socialization of Asian and White men towards the already typical computing identity creates self-reinforcing barriers to diversifying the computing field [20, 43, 59]. Black students face pressure to conform to the dominant culture to become credentialed, aiding in the reproduction of existing norms. Such is the case at the Predominantly White Institutions where most Black undergraduates in the U.S. study [56]. A first step towards an inclusive approach to CS education is understanding how Black students themselves think about identity in computing [59], both at Predominantly White Institutions (PWIs) and HBCUs.

In the U.S., HBCUs, originating from racial segregation, continue to promote the academic

and career success of Black students in the modern day [56]. Being Black at an HBCU is fundamentally different than at a PWI, so it is important to consider the perspectives of students in both settings [15]. An HBCU provides a more culturally affirming environment where Black students are often the majority of the student body [56]. Since there is less pressure to conform to White society at an HBCU, it is a useful site for studying Black identity. One countermeasure to the cycles of cultural reproduction introduced in Section 5.3.1 is counter-storytelling, where a non-dominant racial group can be given a voice to counteract repeated justification for why a dominant racial group holds power in society [19]. Therefore, by studying the perspectives of Black HBCU students, researchers can reshape narratives about who belongs in computing using the voices typically excluded [59].

5.4.2 Persistence

Who participates in computing is largely determined by persistence (successful completion) in undergraduate CS classes and courses of study. Only about two thirds of students enrolled in Introductory Computer Science (CS1) will earn a passing grade [4]. Even if this figure is not remarkable compared to other disciplines, CS has gained a reputation for being difficult and slow to learn [40, 46, 48, 61]. CS educators can work to reshape this perception [46], but doing so requires understanding the factors that help and harm students' abilities to persist.

Four common persistence factors cited in the literature are difficulty, relative deficits, social factors, and applicability. The perception that CS is difficult is the first common reason students avoid or leave CS [2, 6]. When students are socialized and educated differently prior to post-secondary education, deficits in experience, knowledge, and confidence relative to peers create additional barriers to persistence [3, 6, 37]. Since Black students in the U.S. have less access to the best primary and secondary education, they are set up to have deficits entering post-secondary education [68]. Social support [3] from family [14], peers [2], and educators [6] has a strong positive influence on persistence. However, due to cultural reproduction [8] and racial biases in the U.S. legal system [68], Black students are less likely to have close relationships

that foster pursuing a career in computing. A final common influence on persistence is students' perception that the computing skills they learn in school are not translationally applicable in the real world [3, 6, 60]. As discussed in Section 5.3.2, Black students value communal goals, so they are less likely to persist in computing when it is presented as purely individualistic and investigative. Overall, empirical work confirms that identity factors like gender and race can amplify the harms of deficits and social barriers for marginalized students, specifically [2, 3, 20].

5.4.3 AI and Equity in CS Education

Generative AI disrupted traditional CS education around 2022 when it became both widely available and highly performant on basic programming problems [42, 27]. In addition to considering how AI can be used by educators [67], research has begun to explore how novice programmers use code-generating AI-driven Development Environments (AIDEs) such as generic chatbots and in-IDE assistants [39, 41, 54]. While AIDEs can enhance novices' abilities and make programming easier [41, 30], they also threaten to harm students' learning and introduce new cognitive, social, and academic challenges [42, 52, 54].

Despite limited empirical evidence about how to best balance AI-inclusive and AI-exclusive instruction, new pedagogical approaches are already embracing code-generating AI [21, 53]. This presents an urgent need to explore the implications of such AI-inclusive instruction, especially as they relate to identity and persistence in CS. Code-generating AI may be able to lessen some of the persistence barriers discussed in Section 5.4.2, but may also exacerbate inequities by creating new barriers or worsening existing ones [42, 54]. For example, AI provides a private learning environment that can circumvent social barriers, while also reducing beneficial peer and instructor interactions. AI tools with guardrails against code-generation, such as CodeHelp [44], offer a more mild and incremental change to existing pedagogy that helps groups such as non-native English speakers overcome barriers to their success [49]. However, with code-generating options being widely available to novice programmers, students may seek them out to avoid educational guardrails.

A concrete economic challenge is the marginal cost of increased AIDE performance among market offerings; AI companies charge more money for cutting-edge AI technology. Prior disruptions from new technologies (e.g., the internet) have contributed to a “digital divide” [70]. Income and educational disparities will lead students who can easily access the best, most costly AI tools to have better information about their utility compared to students who cannot [42].

Overall, the equity implications of AIDEs are underexplored. A 2023 CS education working group specifically cited an ongoing need for research to address “how these new tools impact equity, justice, and diversity in computing education” [55]. To this end, we set out to explore these implications by collecting the perspectives of Black novice programmers after using the GitHub Copilot AIDE.

5.5 Method

5.5.1 Participants

Ethical authorization was obtained from the Institutional Review Boards of two author-affiliated universities, including an HBCU where participants were recruited. The HBCU’s IRB approved a formal reliance agreement granting ethical oversight to the other university, which approved the study prior to recruitment. With the permission of the college dean, participants were recruited through convenience sampling from all five sections of a second-semester Introduction to Computer Science (CS2) course at the HBCU. These students had all recently completed a Python CS1 course, matching their ability levels well to the tasks in the study. In total, about 100 students were enrolled across these sections and invited to participate through a combination of online and in-person recruitment announcements in February 2024. Between February and May, a gender-balanced sample of 10 participants (5 female; 5 male; 0 other) completed the study. All but one participant had experience using AI chatbots, but none had experience using GitHub Copilot. Participants were compensated \$100 in digital gift cards and offered additional \$10 bonuses to incentivize programming performance; participants were scored based on how

many Python tasks they completed correctly across all trials.

5.5.2 Procedure

The study was run from a university conference room. During the study, participants completed the following: (1) a pre-study questionnaire, (2) Python programming alone and with GitHub Copilot, and (3) an exit interview. The pre-study questionnaire and exit interview were complementary to each other and were adapted from the semi-structured [63] interview guide used by Rankin et al. to investigate intersectional [16] student perspectives in CS [59].

5.5.3 Pre-Study Questionnaire

Participants completed a questionnaire prior to the start of the experimental portion of the study. The purpose was to understand each participant's sense of personal and computing identity along with the factors they deemed helpful or harmful to persisting in CS. They were asked optional questions about race, gender, sexuality, disability status, and computing role models in addition to free-response questions about a sense of personal identity and computing identity (see *supplementary material* for the full set of questions). The questionnaire also asked participants to share any benefits or drawbacks of their identity within computing and what factors they attributed to their ability to persist in computing education. Responding to each question was optional, and participants were told that their answers would be revisited in a later interview.

5.5.4 Programming Tasks

Participants worked on timed Python programming challenges in Visual Studio (VS) Code on a provided Apple iMac. VS Code was also used in their courses, so they were comfortable using it. The study methodology followed that of [30], which used four sets of eight Python programming tasks from the HumanEval dataset¹ [13]. Participants trained extensively on using GitHub Copilot, running and testing Python code, and submitting their tasks. As part of a counterbalanced design,

¹<https://github.com/openai/human-eval>

participants completed two 20-minute sessions of programming alone and two with the assistance of the most current versions of GitHub Copilot (circa 2024; v1.151) and GitHub Copilot Chat v0.11.1. For each session, they completed as many programming tasks as they could (up to eight), working on one task at a time and continuing until their solution to the current task passed all official unit tests. Unlike in [30], they were allowed to use an internet browser; although, they were prohibited from using online AI tools or resources directly related to HumanEval. Within VS Code, they could access pre-written tests, a PDF task prompt, and a skeleton Python function to implement. Participants were instructed and incentivized with bonus compensation to complete as many tasks as possible, using AI as much or as little as needed. For the purposes of this study, the important aspects of the programming tasks were that they were timed, performance-centric, and covered both solo and AI-assisted conditions.

5.5.5 Exit Interview

After completing all four programming sessions, participants stayed for an exit interview. The same researcher conducted all interviews. The final section of the interview (see *supplementary material*) revisited participants' responses to the pre-study questionnaire for an average of about 15 minutes. The purpose was to clarify identity and persistence factors mentioned in the questionnaire and to explore if and how these might relate to code-generating AI. The interviewer first performed member checking [45] to confirm their understanding of the participant's questionnaire responses. Member checking of the manuscript itself was not performed directly, but by one researcher, who teaches at the HBCU, acting as an advocate for the participants with respect to our findings. Next, the interviewer asked how the participant believed AIDEs like the one they just used would impact their success in CS education. They were also asked about whether peer pressure or financial considerations would impact how or how much the participant would use AIDEs. Finally, participants were asked if they had anything else they wanted to share about their identity or how the AI they used related to it.

5.5.6 Data Analysis

Interviews were recorded and transcribed using the Otter² transcription software. A member of the research team manually edited the initial transcripts to ensure their accuracy. Quotes were selected liberally from transcripts using Taguette³ [57]. To counteract selection bias, all major ideas expressed by participants were included at this point. The data were imported to a custom analysis software⁴ to code and organize the 154 quotes.

The analysis process followed hierarchical [63] reflexive thematic analysis (TA). Reflexive TA gave us the latitude to embrace the inherent subjectivity of our method and focus on describing a breadth of ideas rather than over-quantifying our limited data [9]. We seeded an initial codebook deductively based on our three research questions, but used an iterative induction strategy to let the data naturally reshape the structure of the codebook [10, 9, 26].

The first author alone analyzed the data in four passes. A single-rater approach is acceptable at the beginning of reflexive TA, which embraces the inherent subjectivity of qualitative analysis [9]. The goal was to identify the breadth of ideas expressed rather than a reliable and exhaustive model. The first pass involved reading all interview transcripts and referring to any relevant responses from the questionnaire data. Quotes from questionnaires were only used when the interview referred to a questionnaire response that was not verbally reiterated by the participant during the interview. The second pass produced initial codes for all 154 quotes using a variety of coding styles [63, 66] with no particular restrictions. The third pass involved reorganizing the codebook based on themes within the data. The fourth pass polished the codebook by attempting to explain its contours using only direct quotes from participants. These quotes totaled 56 out of 154, representing the 10 participants between two and eight times each. Quotes were selected that were representative of most quotes under each theme. Within the Results and Discussion, quotes are italicized, with paraphrased portions being left outside quotation marks.

To avoid a single-rater bias, the three other authors independently reviewed the data against the codebook. They checked to see that all themes were distinct ideas communicated directly

²Otter is a web-based automated transcription service.

³Taguette is a free and open-source qualitative analysis software for text.

⁴<https://qdb.n.gardella.cc/>; <https://github.com/njg4ne/viewette>

by participants. Next, they reviewed quotes not selected for the results to ensure adequate representation of all major ideas relevant to the research questions. Again, the goal of the TA process was not to achieve a reliable codebook, but rather to represent the breadth of participants' perspectives about identity, persistence, and code-generating AI in CS education.

5.6 Results

We express our results as a story led by the voices of our participants. Diverging from the precise wording of our questions, participants' interpretations led to a rich set of themes that still aligned with our three research questions about identity, persistence, and the expected impacts of AIDEs. Table 5.1 provides an overview of our codebook. We chose not to report prevalence numbers for each theme because we do not want to convey some themes as more important than others.

5.6.1 RQ1: Identity

In this section, we discuss the major aspects of participants' identities that they shared and emphasized as important to them. Table 5.2 shows an overview of our participants. On the pre-study questionnaire, all 10 participants selected only "Black" as their racial identity. Five identified as female, and five identified as male; none identified with any other gender. Most participants disclosed no disabilities or mental health conditions, though one participant reported suffering from Anxiety. While no questionnaire items asked about national origin, six participants disclosed during their interviews that they were international students. Reported national origins included Zambia, Nigeria, Jamaica, and other countries in the Caribbean.

Participants were also asked to describe in their own words the aspects of their identities that were important to them, including, but not limited to, social categories such as race, gender, or nationality. Most chose to describe personal values and traits: the humanity within their approach to computing and the importance of making an impact. Participants shared social identities related to social temperament (e.g. introversion) and mentioned the categories of nationality, race,

Table 5.1: Summary of themes in participants' quotes.

Topic	Theme	Subtheme
Identity (RQ1)		
	Humanity	Curiosity
		Creativity & Craftsmanship
		Diligence
	Impact	Interpersonal
		Societal
	Social Temperaments & Categories	Introversion
		Nationality
		Intersectionality
	Persistence (RQ2)	
Helpful Persistence Factors		
Gratification	Short-term	
	Long-term	
Social Support	Role-models	
	Family expectations	
	Religion	
Self-Teaching		
Financial Aid		
Harmful Persistence Factors		
Difficulty of CS		
Deficits	Access	
	Confidence	
	Knowledge	
Social Barriers	Introversion	
	Pride	
	Collaboration	
	Nationality	
	Underestimation	
AI Persistence Impacts (RQ3)		
Benefits	Overcoming Deficits	
	Moral Support	
	Circumventing Social Barriers	
Drawbacks	Exposing Deficits	
	Promoting Antisocial Tendencies	
	Preventing Learning	
	Disrupting Identity	
	Financial Burden	

Note. Quantification was avoided to align with reflexive TA's focus on breadth of ideas. The topics were predefined from RQs; themes and subthemes arose from ideas reported by participants.

and gender.

Table 5.2: Demographics and backgrounds of participants.

ID	National Origin	Age	Gender	Additional Information	Degree Completion Confidence
j1		18	Male	Extended family role model	Extremely
j11		18	Male	No role models	Somewhat
j13	Jamaica	23	Female	Friend role model	Somewhat
j15	Nigeria	20	Male	Sibling role model	Extremely
j16		18	Male	Friend role model	Very
j2	Zambia	19	Female	No role models	Very
j20	Caribbean	19	Male	Celebrity role model	Very
j24	Caribbean	20	Female	Celebrity role model	Very
j4		18	Female	Parent role model	Very
j5		18	Female	Anxiety, Lesbian sexuality	Extremely

Humanity

Humanity refers to personality traits that take work in computing beyond a pragmatic, productivity-focused mindset to one focused on the experience of the computer scientist as a human. Seven participants (j11, j13, j15, j16, j20, j24, j2) shared that their humanity was an important part of their identity, expressing this through three key values: curiosity, creativity and craftsmanship, and diligence.

Curiosity emerged as participants' desire to learn and experience new ideas. Participant j20 described their inspiration for understanding computing phenomena: *I watched spy movies, but "hacking isn't even the same as what they do. [...] I want to make that realistic for me—like I want to see what it actually looks like."* ^{j20}

Creativity and craftsmanship encompass students' ability to adapt under constraints while applying personal artistry to technical work in computing. When discussing AI tool usage, participant j2 demonstrated confidence in their creative problem-solving abilities: *During the trial when Copilot used the built-in max function, "I wouldn't have thought of the max function [...] I feel*

like because I am creative, I still would have found a way to do it, even if I didn't know the max function.” j²

To further demonstrate the importance of personal aesthetic standards in computing work, participant j24 explained their approach to an assignment: *“Last semester, I was doing a to-do list, which is pretty simple, I guess. But to me, I wanted my code to look uniform. I wanted everything to be catered towards how I would have wanted my to-do lists to be.” j²⁴*

Diligence describes the value participants placed on working hard, embracing challenges, and pushing through adversity. Two responses exemplify this commitment to persistence: *“I believe I can solve computing problems if I try hard enough, no matter how long it takes.” j²* and *“I enjoy learning hard things.” j¹⁵*

These quotes demonstrate that participants have interest and recognition of their abilities, key aspects of computing identity [33]. They find this identity not through a focus on speed and performance but from savoring the programming process.

Impact

Impact refers to participants' solidarity with both the people around them and the members of society who may use or benefit from the software they create. Four participants (j11, j13, j15, j20) shared that their identity was rooted in a desire to make such an impact in the world.

To illustrate how impact manifests in peer relationships, participant j11 described their response to academic struggles: *I've struggled with assignments, “and when people came to me asking for help, I gave them the best advice I could for them, so they don't end up in the same mode [as me].” j¹¹*

Moving from peer support to broader societal contribution, participant j13 explained their motivation for pursuing CS: *“One of the main reasons I'm doing computer science is it's all pretty new and it's innovative [...] it's very interesting discovering new things and then contributing to stuff or making it more improved.” j¹³*

This response shows how students connect their individual learning journey to broader technological progress and social improvement, meaning that they are positioning themselves as

agents of positive change through computing.

Social Temperaments and Categories

We combine social temperaments (e.g. introversion) with social categories (e.g. gender) as they both represent arbitrary labels society uses to reinforce existing power structures. From a Bourdiesian perspective, labels like “introvert” can artificially justify why someone without a respected title (social capital) deserves limited power [7]. Similarly, awareness of one’s atypical gender or race in some spaces can prevent them from questioning exclusion disguised as meritocracy [8]. Four participants (j12, j24, j4, j2) mentioned their social temperament as a part of their identity, with one identifying as an ambivert, while the others described themselves as shy or introverted. However, given our international student sample, national origin emerged as the most important social category. To demonstrate the importance of national identity in shaping computing aspirations, a Nigerian participant provided a poignant example. While describing the technology culture in their city, they reflected on the humble beginnings of companies like Paystack⁵, a successful Nigerian financial technology startup. *“you’ll be able to hear like stories that are similar to yours. You know, ‘Oh, this guy just came up from nothing [...] being able to like raise such and such an amount of money for his startup’ [...] I was really inspired by stuff like that.”* j20

The only mention of race or gender was from participant j5, who shared a complex point about her intersectional (Black and female) identity in computing. She questioned a narrative she had heard from others. Despite believing it to be false, she seems to have partially internalized it. *“Diversity hires are a thing. I can’t deny that it is. And I’ve heard from some people that being a woman in the computer science field, you can have lower skill but get hired at jobs that are above your skill level, which I don’t think is true. I think it’s just angry men saying that.”* j5

Students from underrepresented groups must simultaneously reject discriminatory narratives while acknowledging their presence in professional computing spaces.

⁵<https://paystack.com/>

5.6.2 RQ2: Persistence

Helpful Persistence Factors

We asked participants about what factors they attributed to their ability to succeed in CS and what benefits they perceived related to having their identity within computing. Overall, gratification was the greatest motivator for persistence, followed by social support, financial aid, and self-teaching.

Gratification Gratification refers to rewards such as fun, praise, academic achievement, financial gain, or status advancement. Eight participants (j11, j13, j15, j16, j20, j24, j4, j5) indicated this factor as important for their persistence, encompassing both immediate and future rewards. Short-term gratification emerged through achievements including succeeding in CS classes and creating real products with coding. To demonstrate how early success builds sustained motivation, participant j15 explained their initial coding experience: *First learning to code “was really exciting, especially, like, being able to like create stuff [...] I was very interested in it, so I felt like motivated to keep going at it.”* ^{j15}

This sense of achievement has a lasting impact as does completing one’s first CS course, as participant j16 describes: *Getting through CS1 “definitely built my confidence a lot, and then even now, like, I feel proud of myself.”* ^{j16}

Future gratification motivated students through anticipated professional, financial, and innovative outcomes. Participants considered the prestige of a CS degree: *“For myself, I really do want a degree, and I don’t want just any degree; I want it to be in STEM.”* ^{j24}

This is accompanied by acknowledging the financial drivers: *“One thing that keeps me pursuing a computer science degree is the average salary.”* ^{j5}

The possibility of creating innovative products offered a third source of future gratification. One student from Jamaica connected their impact aspirations to invention potential: *The idea that I can make an impact “does keep me going with [CS]. Who knows? I might invent something one day.”* ^{j13}

Social Support Participants were asked directly about their computing role models (see *supplementary material*), and several participants shared in their interview how the role models they identified were helpful to their success in computing. However, some mentioned other social structures: two (j20, j24) described family expectations about their success and j13 shared the importance of their religion. In addition to the founder of Paystack being a celebrity role model for participant j20 (see Section 5.6.1), participants identified role models in family (j15, j24), friends (j13, j16, j24), and professors (j24). *I think my brother “is currently working at Microsoft, so I was really inspired by that, and that really pushed me [...] to, like, study hard and get here so I can keep like pushing my goal to be, like, a really good software engineer.”* ^{j15}

In addition, a student from Nigeria empathizes: *“Seeing [my friends] put in the work and be really good at it makes me want to do the same thing.”* ^{j16} One student from the Caribbean described connecting with a professor’s background through shared identity and overcoming similar challenges: *One of my professors “told me about his struggles being an international student and learning to code, English not being his first language. So, I really wanted to like, be like him, you know, in the field of CS.”* ^{j24}

Self-Teaching Finally, three participants (j1, j20, j24) emphasized the importance of pursuing extracurricular learning through self-teaching. Participant j1 explains: *“I did a lot of self-teaching before [the year] started, so I was able to already be done with all the assignments for about five weeks into the program. [...] it makes it easier and less stress, less pain, less feelings like that.”* ^{j1} A student from the Caribbean explains: *“I like to explore, like, I haven’t even learned machine learning inside any of my classes yet, and I’m doing machine learning projects. So that’s basically me learning on my own.”* ^{j20}

Financial Aid While financial aid was not mentioned often, two participants (j5, j20) said that scholarships and grants enabled them to succeed. Participant j5 directly acknowledged financial support necessity: *“I’ve got like grants and scholarships, and things like that have helped me be in school.”* ^{j5} Participant j20 connected scholarship receipt to self-perception: *“When I came*

here, I got a full scholarship. And then, like, I just realized that I'm sort of good at this, you know, so it's like, why don't I just stick with this?" j20

Financial aid can determine educational opportunities. This financial recognition goes beyond mere economic support and can function as ability validation for underrepresented students.

Harmful Persistence Factors

We asked participants about barriers related to their identities in computing but other types of barriers emerged across interviews. For this reason, this section focuses on all types of barriers and not just those related to identity. These harmful persistence factors included difficult CS coursework, deficits relative to peers, and social barriers.

Difficulty Five participants (j1, j11, j13, j16, j2) said that the difficulty of the work in a CS curriculum was a barrier to success: *CS is difficult. "Oh, yeah, absolutely [...] I'm managing but it's still very, very hard, very difficult."* j16

Participant j2, from Zambia, felt discouraged when they believed they should have been prepared to understand computing concepts but still could not. *"It's just discouraging when you're not understanding something [...] it made me feel dumb for not understanding [...] sometimes it makes me want to give up."* j2

Even participant j1, who found success through self-teaching, admitted that the work was too tiring to sustain. *I was ahead from self-teaching, but I stopped doing as much. "It was just, to be honest, it was just, exhausting, just tired of working."* j1

Deficits In addition to finding coursework hard, many participants felt that they were at a disadvantage based on unequal deficits they had compared to others. Five participants (j13, j15, j16, j20, j24) described how deficits in their skills, knowledge, experience, confidence, or access to computing harmed their ability to succeed. For example, participant j15 had no access to a computer of their own during secondary education: *"When I learnt about like programming and stuff like that, I really wanted to do it, but I didn't have a laptop, so any opportunity that I could*

use, I would try to, like, use it to code. That included me borrowing my mom's phone to, like, just learn when she was sleeping and stuff like that. And sometimes like after school, I would go to a cybercafé." j15

Participant j16 mentioned how a lack of access to computer science education prior to college also caused a lack of confidence in their abilities: *I definitely did not have much confidence "coming to college, period, especially since I hadn't touched a piece of code before I got here [...] I was very, very unconfident in myself."* j16

Finally, participant j24 captured why deficits can be so damaging to students. They mentioned a passion for graphic design and a desire to become a front-end developer but chose to lean into a secondary major instead based on deficits relative to peers in CS. Worse yet, they implied that negative comments from peers informed their surrender: *I am focusing more now on data analytics. "So, my classmates won, I guess. I still like the idea of being a front-end developer, but it doesn't always work out. [...] As a computer science student, you're expected to have a certain knowledge. And when you feel like you don't have that knowledge, you feel pretty, you know, behind everybody else. So you don't really want them to know because, you know, not everybody's very nice."* j24

Social Barriers Five participants (j13, j20, j24, j4, j2) mentioned social factors that could harm their success in computing. Social barriers include external barriers to support as well as internal psychological factors that prevent students from accessing available social resources. These included challenges with social structures in computing due to independent working style, pride, introversion, cultural differences, citizenship status, and having their abilities underestimated. These comments were often linked back to participants' experiences interacting with recruiters at university career fairs. Two participants (j5 and j24) found that social anxiety and introversion have prevented them from accessing valuable social experiences within their education: support with coursework (j5) and participating in hackathons (j24)—*"I struggle with social anxiety which makes it hard for me to get help from my professors or tutors."* j5

On the other hand, participants j13 and j20 had barriers accessing social support for course-

work because of pride (j13) and preference for self-reliance over collaboration (j20), which prevented them from utilizing available collaborative resources, eventually hindering them to build a network. *I don't always ask for help even when I'm stuck. "People always say Python is easy. So, well, if it's easy, I can do it! But like, then, it's not that easy. So, I guess maybe it's a little bit of pride too."*^{j13} One student from the Caribbean shares: *"One of my mentors is like, 'it's not always about what you know; it's about who you know', and me I really don't like group projects when it comes to programming. [...] if somebody isn't able to do something, I can't handle being able to like, pick up their stuff."*^{j20}

Other challenges arose for international students. Participant j24 struggled to fully engage in classes due to accent-related embarrassment, while students j13, j24, and j4 were prevented from industry experience by their citizenship status. *When talking to employers, "I found out that [...] everything in this country like requires some kind of documentation: social security number or be a citizen, or work authorization. All those things, I have no idea about."*^{j4}

A final barrier appeared in the subtext of the comments of two participants about talking to recruiters at a job fair as first year undergraduates. It is not fully clear why this occurred, but it seems as if recruiters at these job fairs were underestimating these students' abilities. Though the students were only in their first semester at the time, they were nonetheless technically competent students. However, the recruiters apparently did not expect them to have technical abilities beyond basic computer skills: *The employers who would talk to me as a freshman would say, "Oh yeah, we need computer scientists [...] like we need someone to go on Word", and I'm like, that's not computer science! That's like basic computer skills."*^{j13} and *"Most of the employers will say that they don't really expect for us to be very, like, excellent in the coding skills"*^{j4}

5.6.3 RQ3: Impact of AI on Persistence Factors

We directly asked participants how they believed AI would impact their abilities to succeed in CS education. They also brought up AI when talking about different helpful and harmful persistence factors. We separated their comments based on whether they cited benefits or drawbacks of

AIDEs, with some overlapping themes.

AI Benefits

Participants anticipated three primary benefits of AIDEs: overcoming deficits, providing moral support, and circumventing social barriers.

Overcoming deficits emerged as a key benefit, with five participants (j13, j16, j20, j24, j2) believing that AIDEs could help them to address knowledge and confidence gaps. For example, participant j13 said Copilot can let them build confidence in private so that they want to participate more in class. *Some of Copilot's responses were so simple. "It really shows [...] it's not that difficult. You just have to really work through the problem [...] [Copilot] makes me [want to] participate more in class, because I'm like, 'Okay I do know what I am talking about'."* ^{j13}

Participant j2 captured how learning from an AIDE could even potentially be better than learning from a human instructor. They say that human instructors are limited to their own knowledge and experiences, whereas AIDEs provide broad perspectives that can go beyond what is taught by a given instructor: *I think it's "extremely important" to expose your own skill deficits. "The thing with the AI is that it's right there, so we can get, like, everything at once [...] if I went to a professor who has taught me the same functions all the time, I wouldn't figure out that other new function because they also know a limited amount."* ^{j2}

Moral support provided psychological comfort through AIDE presence. Four participants (j16, j20, j24, j5) valued the reassurance of this assistance: *"It'll, number one, help my confidence in CS, since it'll be providing me assistance [...] and then it will just make things [...] a little easier for me [...] take a little bit of weight off my shoulders [...] even if you don't use it, like, that 'just in case' you need a little bit of assistance and you can't go get it, you know, Copilot is right there on your shoulder ready to help you."* ^{j16} and *AI would help me not to give up so easily on independent projects, "if I ever get stuck on something, and it's hard for me to find information on that specific thing that I'm doing then I'd consider using AI for it."* ^{j5}

AI helped circumventing social barriers by offering private, judgment-free, pseudo-social assistance environments. Four participants (j13, j20, j24, j2) valued AI's social neutrality. Participant

j24 contrasted AI and human interactions by this judgment-free concept: *“the AI offers less judgment [...] you’re not sitting wondering, like, ‘Do they think I’m stupid or something?’ [...] It’s easier to like just, you know, go all out with an AI versus a human being.”* j24

In addition, AI offered straightforward communication and independent problem-solving without social obligations. Participant j20 appreciated AI’s directness: *It’s pretty easy to communicate with AI, whereas with people “you give them a prompt and they will question the prompt.”* j20

Participant j13 described how AI assistance felt more autonomous: *“Sometimes you, like, you don’t want to bother people, or you don’t want to like, per se, like copy their work, too. So, like, it feels a bit more independent working with the AI.”* j13

AI Drawbacks

Participants anticipated five primary drawbacks of AIDEs: exposing their deficits, promoting antisocial tendencies, preventing learning, disrupting identity, and imposing financial burdens.

Exposing deficits is necessary to overcome them, but Copilot’s high performance made two participants (j11, j2) feel exposed: *Using Copilot makes me think, “Am I even able to do this? Like, could I keep up with this for three years, knowing that I’m going to have to learn a lot of stuff to get this degree?”* j11 and *“In the future [Copilot] would be positive because I feel like I will be learning. I only feel it is negative right now because I can’t figure [the problems] out so it made me feel dumb.”* j2

As with deficits, the social effects of AIDEs can be both helpful and harmful. Two participants (j5, j24) felt that pseudo-social AIDE interactions were a worse alternative to true social interactions. *“I don’t think that AI will replace the kind of help that an actual person can give you. [...] with AI, there’s like no actual emotional connection to it, so I feel like human interaction will just be more rememberable.”* j5

Two other participants (j16, j20) thought that reliance on Copilot might harm their ability to learn if they were not careful: *“it can also hinder my learning [...] if I don’t use it in an educational way, like, I just use it just to get the answer.”* j16

Identity disruption threatened core aspects of participants’ CS self-concepts described earlier

in Section 5.6.1 due to AI's presence. Three participants (j11, j15, j24) worried about AI undermining their humanity and impact identities. Participant j11 felt that AI would harm their strong work ethic, a way they expressed their humanity within their work, and their role as a leader, a way they made a social impact. *AI might discourage my work ethic. "a lot of the times I just do homework and when I feel like I can't do it, that's when I have to rely on ChatGPT."* I might also feel less important helping classmates *"because it's AI; they know a lot."* ^{j11}

Participant j24 thought that Copilot limited their ability to solve problems creatively, one part of the humanity identity. *"I think it sort of limited my creativity. Yeah, just finding different ways to find the solution to a question, because I just, you know, copy and pasted."* ^{j24}

Imposing financial burdens created access inequality concerns. When asked about paying for Copilot, all participants essentially answered that doing so was a personal choice, just like any other financial decision. Participants overwhelmingly (7/10) felt that Copilot was not worth paying for in light of free alternatives like ChatGPT. This creates a clear financial burden in that accessing the best AI is expensive enough to be worthwhile only to *"rich people"* ^{j5}. Participants said that Copilot would be out of reach for them if they could not get free access to it: *If the free trial really impresses me, I'll pay for it, "but if it comes to like, 'Oh, you have to pay to use,' then I'll probably never [do it]."* ^{j15}

5.7 Threats to Validity

Sample size, sampling method, and demographic representation are our primary limitations. Our sample size of 10 participants is too low to reach theoretical saturation [5], and it is difficult to claim that our findings are general, though they are useful for giving underrepresented students a voice. We used convenience sampling for a long study requiring multiple steps and scheduling considerations for participants. This threatens the generality of our findings because participants who were able to complete this study may have had advantages relative to other students at the HBCU. For example, students who also held jobs may have faced unique barriers in CS education [29], yet may not have had free time to participate. To mitigate this limitation, we of-

ferred an especially high level of compensation for a study of this type and provided a long (three month) recruiting window. In terms of demographics, we had a representative split of five men and five women, but no other genders were represented. Furthermore, all participants identified as Black, so Hispanic, Latinx, Indigenous, and Pacific Islander students were not represented. The greatest limitation was the national origin of participants. Six of 10 participants reported a non-U.S. national origin, so our findings are likely skewed towards the perspectives of international students as compared to Black students of U.S. origin [38]. Coming from countries with Black majorities, these participants may not have been forced to consider their racial category throughout their upbringing in the same way as marginalized Black students from the U.S.

5.8 Discussion

5.8.1 RQ1: Identity

The HBCU CS students we interviewed understood their identities in terms of their humanity, their social impact, and their social temperaments and categories. Computer scientists in the U.S. are educated for an economy that incentivizes productivity in service of accumulating wealth [65]. CS education also prioritizes efficiency so that students can finish their education quickly and be effective contributors in the workforce. However, our results reveal that students' identities value the humanity of creativity, craftsmanship, and diligence rather than only the software products themselves. We also found that social impact was an identity-driven motivator for students. This aligns with the idea of social solidarity, through which people act to uphold social harmony beyond what rules require [24]. For example, the leadership (impact) identity we found upheld social cohesion but was not academically required. Student desires to impact society at large by contributing useful software reflect solidarity in the "organic" [24] sense, whereby individuals cope with socially disconnected capitalist economic systems by relying on each other socially by contributing labor to the economy. Self-Determination Theory and the Culturally Engaging Campus Environments model suggest that such social connection is an inherent drive and is needed

for diverse students to succeed [18, 50]. In line with prior work that blames persistence problems on the lack of applicability of CS coursework [3, 6, 60], we found that students need to see the individual and social relevance of CS to find their identity in the discipline. Finally, we found that social temperaments and categories were ways students understood their identities. Social learning entails integrating personal identity with community norms [71], so an established computing community lacking diversity discourages participation from both shy students and those in marginalized categories [23, 59, 62]. On the other hand, seeing someone of your own nation create a successful startup, for example, is inspirational and encouraging [3].

5.8.2 RQ2: Persistence

We found that the difference between students' preparation for CS education and the difficulty of the coursework was harmful, while positive feedback loops were helpful. Echoing prior work, some participants perceived the workload in CS to be challenging, exhausting, and discouraging [2, 6]. Facing this work was especially challenging, though, for those who lacked confidence, computer access, or computing education prior to their post-secondary studies. Prior research confirms that such deficits place these students on unequal footing [3, 6, 37]. Aligning with social learning theory [71] and prior studies, social barriers also hindered students' success, while social support encouraged it [2, 3, 6, 14]. In addition to positive social feedback loops, gratification cycles motivated students through both small accomplishments and hope for future rewards. An effective technique for creating short-term gratification is gamification, which is notably enhanced by competition through leaderboards [32]. Gari et al. acknowledge that competition has the downside of emphasizing extrinsic motivation over intrinsic motivation [32]. Self-Determination Theory (SDT) argues that people are better at facing difficult activities when intrinsically motivated [18]. Competition may also introduce negative feedback by emphasizing deficits compared to peers, so gamification may be more helpful in the context of individual work and self-teaching. As for long term gratification, simple knowledge of a CS degree's potential may be sufficient to encourage persistence. Finally, it must be acknowledged that financial aid programs, such as those

mentioned by our participants, are an important part of diversifying CS. Race is not a proxy for income, but the median Black household income was still 30% below that of all U.S. households in 2022 [34]. Until this income gap is closed, grants and scholarships will continue to promote representation of Black students in computing.

5.8.3 RQ3: AI Impact on Identity and Persistence

We found that AIDEs can promote persistence by providing pseudo-social support that addresses students' deficits; however, they can harm persistence by disrupting personal and social identities and encouraging dependence on AI over superior social supports. Participants felt that GitHub Copilot made them feel not just more efficient as in [41, 54, 30], but morally supported. It also gave them a private learning space to work out their knowledge gaps and insecurities in ways they could or would not with humans. Prior work suggests that AIDEs will widen gaps between strong and struggling students [41, 54], but our findings show how AIDEs could help close those gaps instead. On the other hand, in line with prior work we found that AIDEs can encourage AI dependence that harms self-efficacy [47, 54]. The convenience of AIDEs will prevent students from accessing support from family, peers, mentors, and instructors, which is known to promote persistence. Free AIDEs will be most accessible but least suitable as social alternatives, creating a new "digital divide" [70] in the age of AI. Even though participants were given knowledge about the utility of a premium AIDE within the study, they could not justify its cost. In conclusion, omnipresent AIDEs may distance CS students from their personhood and social worth in the computing community. Nonetheless, banning AIDEs in CS education is clearly neither feasible nor advisable.

5.9 Recommendations

5.9.1 AI-Agnostic Recommendations

The persistence factors we encountered argue for three non-AI interventions to promote persistence of diverse students in CS.

First, educators should look for ways to gamify the student experience. For example, students can compete on low-stakes in-class programming challenges in the style of the popular study game Kahoot⁶. To keep motivation intrinsic, students can also be encouraged to compete against themselves individually. For example, a student could track their daily completion of lessons on freeCodeCamp⁷, aiming to complete more lessons per day than their usual. Students are motivated by their incremental achievements, but their efforts do not always yield visible progress [48]. Several of our participants said they had little experience with automated tests like the ones in this study; educators can encourage students to create their own progress indicators using unit testing [25]. Other ways to reward diverse students are to recognize them with awards and selection for opportunities such as teaching assistantships, internships, fellowships, and academic presentations.

Next, educators should strengthen social interventions. Given that students identified social support as crucial for persistence while simultaneously reporting social barriers as major obstacles, computing educators must intentionally design social interventions. These might include encouraging faculty mentorship and promoting beneficial peer interactions like pair programming [35].

Lastly, investment in financial aid should be prioritized to encourage students from low-income families to pursue CS. With participants directly citing financial constraints as both success promoters (through aid) and detriments (through lack of resources), computing education programs must advocate for financial support. This might be implemented by creating paid opportunities such as undergraduate research positions and teaching assistant roles.

⁶Kahoot is a multi-round interactive quiz game with a leaderboard.

⁷freeCodeCamp is a free online app with interactive programming courses.

5.9.2 AI-Based Recommendations

Regarding the use of AIDEs in CS education, we recommend a conservative deployment strategy.

We found that students felt AIDEs would support them and help them overcome primary barriers to persisting in CS, so fully prohibiting AIDEs misses an opportunity to use this new technology for a higher social good. However, we see the omnipresence of AIDEs in educational approaches such as [53] to be overall detrimental to equity in CS. Instead, educators should consider including AIDEs intentionally for certain assignments, while restricting their use on others using proctoring. An alternative would be to restrict AIDE use in most cases, while creating exceptions for struggling students and students with formal social or emotional disability accommodations. This way, AIDEs could be used to close achievement gaps in CS courses instead of widening [54] them. For shy students without severe anxiety, educators might consider social interventions like pairing friends to program together instead. Finally, instructors should teach students to customize AIDEs to their needs [54]. Providing students with restrictive AI like Code-Help [44], though promising, may lead students to seek outside alternatives without restrictions. Instead, we recommend providing students agency to restrict AIDEs themselves. New AIDE offerings (e.g. Cursor⁸) allow for “system prompts” or “rules” that modify how they behave across all responses. Students can customize these prompts in ways like “provide me hints on how to solve a problem before showing me the code,” “prefer readability over conciseness,” or “talk to me like a beginner who only knows Scratch.”

Finally, colleges and universities should invest in corporate partnerships that allow students to use premium AIDEs for free during their education. An example of this in action is the GitHub Education program, which currently gives students free access to the premium version of GitHub Copilot.

⁸Cursor is a fork of the open-source version of VS Code that added AI-driven code editing features not previously possible with Copilot in VS Code (but which have since been added to Copilot, too).

5.10 Conclusions and Future Work

In this case study, we used qualitative data to reveal the perspectives of 10 Black CS students studying at an HBCU around the nexus of identity, persistence, and code-generating AI. Inclusive CS education must be a gratifying, social, and financially feasible experience for students before considering AI. However, we argue that conservative and deliberate deployment of code-generating AI within CS education can help to close achievement gaps. Indiscriminate deployment of these AI tools in computing stands to devalue human programmers and deprioritize their well-being. Overall, our findings suggest that AIDEs cannot currently replace the inherently social nature of an inclusive computing discipline.

Future work can consider extending our work to larger samples, samples with more students of U.S. origin, and samples of other marginalized groups (e.g. Hispanic students). Our scope precluded a discussion on the need for students to become effective AI users, but researchers should continue [21] to explore this area. The thread of social learning throughout our work presents a critical need to further explore (1) what social interventions best promote CS students' success and (2) how AIDEs can augment social learning.

Acknowledgments

Thank you to Stacey Williams from Virginia State University for her help with the logistics of this study.

References

- [1] Computing Research Association. 2017. Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006. (2017).
- [2] Lecia J. Barker, Charlie McDowell, and Kimberly Kalahar. 2009. Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin*, 41, 1, (Mar. 2009), 153–157. Retrieved Sept. 25, 2023 from.

- [3] Alvine B. Belle, Callum Sutherland, Opeyemi O. Adesina, Sègla Kpodjedo, Nathanael Ojong, and Lisa Cole. 2023. Bolstering the Persistence of Black Students in Undergraduate Computer Science Programs: A Systematic Mapping Study. *ACM Transactions on Computing Education*, 23, 4, (Dec. 2023), 1–42. Retrieved Mar. 18, 2025 from.
- [4] Jens Bennedsen and Michael E. Caspersen. 2019. Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10, 2, (Apr. 2019), 30–36. Retrieved July 16, 2025 from.
- [5] H. Russell Bernard, Amber Wutich, and Gery W. Ryan. 2016. *Analyzing Qualitative Data: Systematic Approaches*. SAGE publications. Retrieved July 26, 2024 from.
- [6] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. 2008. Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. *ACM SIGCSE Bulletin*, 40, 1, (Mar. 2008), 402–406. Retrieved Sept. 25, 2023 from.
- [7] Pierre Bourdieu. 2011. The Forms of Capital. In *Cultural Theory: An Anthology*. Vol. 1, 949. Retrieved July 8, 2025 from.
- [8] Pierre Bourdieu and Jean Claude Passeron. 1977. *Reproduction in Education, Society and Culture*. Sage studies in social and educational change. Sage Publications, London, Beverly Hills. isbn: 978-0-8039-9994-7 978-0-8039-9995-4.
- [9] Virginia Braun and Victoria Clarke. 2022. Conceptual and design thinking for thematic analysis. *Qualitative psychology*, 9, 1, 3. Retrieved July 12, 2024 from.
- [10] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3, 2, (Jan. 2006), 77–101. Retrieved Oct. 2, 2023 from.
- [11] U.S. Census Bureau. 2024. QuickFacts: Age and Sex. <https://www.census.gov/quickfacts/fact/table/US/PST045224>. (2024). Retrieved Mar. 18, 2025 from.
- [12] U.S. Census Bureau. 2024. QuickFacts: Race and Hispanic Origin. <https://www.census.gov/quickfacts/fact/table/US/PST045224>. (2024). Retrieved Mar. 18, 2025 from.
- [13] Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*. arXiv: 2107.03374.
- [14] Jody Clarke-Midura, Frederick J. Poole, Katarina Pantic, Chongning Sun, and Vicki Allan. 2018. How Mother and Father Support Affect Youths' Interest in Computer Science. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM, Espoo Finland, (Aug. 2018), 215–222. isbn: 978-1-4503-5628-2. Retrieved Dec. 28, 2023 from.

- [15] Kevin Cokley, Ramya Garba, Keoshia Harris, Nolan Krueger, Marlon Bailey, and Shaina Hall. 2023. Student-Faculty Interactions, University Environment, and Academic Attitudes Among Black College Students: The Role of School Racial Composition. *The Journal of Higher Education*, 94, 4, (June 2023), 444–472. Retrieved July 15, 2025 from.
- [16] Patricia Hill Collins. 2002. *Black Feminist Thought: Knowledge, Consciousness, and the Politics of Empowerment*. (2nd ed.). Routledge, New York, (May 2002). isbn: 978-0-203-90005-5.
- [17] Cristobal de Brey, Kristi Donaldson, NCES Supervisors, and NCES Annual Reports and Information Staff. 2024. 2022 Digest of Education Statistics. Tech. rep. Institute of Education Sciences- United States National Center for Education Statistics, Washington, DC, USA, (Feb. 2024). Retrieved Mar. 18, 2024 from.
- [18] Edward L. Deci and Richard M. Ryan. 2013. *Intrinsic Motivation and Self-Determination in Human Behavior*. Springer Science & Business Media, (June 2013). isbn: 978-1-4899-2271-7.
- [19] Richard Delgado and Jean Stefancic. 2023. *Critical Race Theory: An Introduction*. Vol. 87. NyU press. Retrieved July 15, 2025 from.
- [20] Jennifer Dempsey, Richard T. Snodgrass, Isabel Kishi, and Allison Titcomb. 2015. The Emerging Role of Self-Perception in Student Intentions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, Kansas City Missouri USA, (Feb. 2015), 108–113. isbn: 978-1-4503-2966-8. Retrieved Mar. 17, 2025 from.
- [21] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, (Mar. 2024), 296–302. isbn: 979-8-4007-0423-9. Retrieved Mar. 19, 2024 from.
- [22] Amanda B. Diekman, Mia Steinberg, Elizabeth R. Brown, Aimee L. Belanger, and Emily K. Clark. 2017. A Goal Congruity Model of Role Entry, Engagement, and Exit: Understanding Communal Goal Processes in STEM Gender Gaps. *Personality and Social Psychology Review*, 21, 2, (May 2017), 142–175. Retrieved July 8, 2025 from.
- [23] Remy Dou, Karina Bhutta, Monique Ross, Laird Kramer, and Vishodana Thamotharan. 2020. The Effects of Computer Science Stereotypes and Interest on Middle School Boys' Career Intentions. *ACM Transactions on Computing Education*, 20, 3, (Sept. 2020), 1–15. Retrieved Mar. 18, 2025 from.
- [24] Emile Durkheim. 2023. The Division of Labour in Society. In *Social Theory Re-Wired*. (3rd ed.). Routledge. isbn: 978-1-00-332060-9.

- [25] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: automatically grading programming assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*. ACM, Madrid Spain, (June 2008), 328–328. isbn: 978-1-60558-078-4. Retrieved Mar. 20, 2025 from.
- [26] Dr Nigel G. G. Fielding, Nigel Fielding Raymond M. Lee, and Professor Raymond M. Lee. 1998. *Computer Analysis and Qualitative Research*. SAGE, (June 1998). isbn: 978-1-4462-2815-9.
- [27] 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*. ACM, Virtual Event Australia, (Feb. 2022), 10–19. Retrieved July 21, 2025 from.
- [28] Richard Fry, Brian Kennedy, and Cary Funk. 2021. STEM jobs see uneven progress in increasing gender, racial and ethnic diversity. *Pew Research Center*, 1–28. Retrieved Sept. 25, 2023 from.
- [29] Susan R. Furr and Theodore W. Elling. 2002. African-American students in a predominantly-White university: Factors associated with retention. *College Student Journal*, 36, 2, 188–203. Retrieved Mar. 19, 2025 from.
- [30] Nicholas Gardella, Raymond Pettit, and Sara L. Riggs. 2024. Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ACM, Milan Italy, (July 2024), 290–296. Retrieved July 21, 2025 from.
- [31] Nicholas Gardella, Joseph Shelton, Isabella Graßl, and Sara Riggs. 2025. HBCU student perspectives on identity, persistence, and code-generating AI in CS education: a case study. In *Proceedings of the 25th Koli Calling International Conference on Computing Education Research*. Koli Calling '25: 25th Koli Calling International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 11, 2025), 1–12. isbn: 979-8-4007-1599-0. doi:10.1145/3769994.3770015.
- [32] Mourya Reddy Narasareddy Gari, Gursimran Singh Walia, and Alex David Radermacher. 2018. Gamification in Computer Science Education: a Systematic Literature Review. In *2018 ASEE Annual Conference & Exposition*. (June 2018). Retrieved Mar. 20, 2025 from.
- [33] Gregor Große-bölting, Dietrich Gerstenberger, Lara Gildehaus, Andreas Mühling, and Carsten Schulte. 2023. Identity in Higher Computer Education Research: A Systematic Literature Review. *ACM Trans. Comput. Educ.*, 23, 3, (Sept. 2023), 35:1–35:35. Retrieved June 24, 2025 from.
- [34] Gloria Guzman and Melissa Kollar. 2023. Income in the United States: 2022. Tech. rep. P60-279. U.S. Census Bureau, Washington, DC, (Sept. 2023). Retrieved Mar. 20, 2025 from.

- [35] Anja Hawlitschek, Sarah Berndt, and Sandra Schulz. 2023. Empirical research on pair programming in higher education: a literature review. *Computer Science Education*, 33, 3, (July 2023), 400–428. Retrieved Oct. 5, 2023 from.
- [36] John L. Holland. 1985. *Making Vocational Choices: A Theory of Vocational Personalities and Work Environments*. Prentice-Hall, Englewood Cliff, NJ. isbn: 978-0-13-547597-3.
- [37] Diane Horton and Michelle Craig. 2015. Drop, Fail, Pass, Continue: Persistence in CS1 and Beyond in Traditional and Inverted Delivery. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, Kansas City Missouri USA, (Feb. 2015), 235–240. isbn: 978-1-4503-2966-8. Retrieved Sept. 27, 2023 from.
- [38] Kathleen Isenegger, Kari L. George, Paul Bruno, and Colleen M. Lewis. 2023. Goal-Congruity Theory Predicts Students' Sense of Belonging in Computing Across Racial/Ethnic Groups. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM, Toronto ON Canada, (Mar. 2023), 1069–1075. Retrieved July 8, 2025 from.
- [39] 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*, 1–15.
- [40] Tony Jenkins. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. Vol. 4. Citeseer, 53–58. Retrieved Nov. 7, 2023 from.
- [41] 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, (Apr. 2023), 1–23. isbn: 978-1-4503-9421-5. Retrieved June 21, 2023 from.
- [42] Sam Lau and Philip 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*. ACM, Chicago IL USA, (Aug. 2023), 106–121. isbn: 978-1-4503-9976-0. Retrieved Oct. 3, 2023 from.
- [43] Kathleen J. Lehman, Linda J. Sax, and Hilary B. Zimmerman. 2016. Women planning to major in computer science: Who are they and what makes them unique? *Computer Science Education*, 26, 4, (Dec. 2016), 277–298. Retrieved Mar. 18, 2025 from.

- [44] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. ACM, Koli Finland, (Nov. 2023), 1–11. isbn: 979-8-4007-1653-9. Retrieved June 23, 2025 from.
- [45] Yvonna S. Lincoln and Egon G. Guba. 1985. *Naturalistic Inquiry*. Vol. 75. Sage. Retrieved Aug. 28, 2024 from.
- [46] Andrew Luxton-Reilly. 2016. Learning to Program is Easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, Arequipa Peru, (July 2016), 284–289. isbn: 978-1-4503-4231-5. Retrieved Nov. 7, 2023 from.
- [47] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-Regulation, Self-Efficacy, and Fear of Failure Interactions with How Novices Use LLMs to Solve Programming Problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ACM, Milan Italy, (July 2024), 276–282. isbn: 979-8-4007-0600-4. Retrieved Nov. 22, 2024 from.
- [48] Michael McCracken et al. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (ITiCSE-WGR '01). Association for Computing Machinery, New York, NY, USA, (Dec. 2001), 125–180. isbn: 978-1-4503-7359-3. Retrieved Sept. 27, 2023 from.
- [49] Ismael Villegas Molina, Audria Montalvo, Benjamin Ochoa, Paul Denny, and Leo Porter. 2024. Leveraging LLM Tutoring Systems for Non-Native English Speakers in Introductory CS Courses. (Nov. 2024). Retrieved July 8, 2025 from arXiv: 2411.02725 [cs].
- [50] Samuel D. Museus. 2014. The Culturally Engaging Campus Environments (CECE) Model: A New Theory of Success Among Racially Diverse College Student Populations. In *Higher Education: Handbook of Theory and Research*. Dordrecht. isbn: 978-94-017-8004-9 978-94-017-8005-6.
- [51] Anne-Kathrin Peters. 2014. *The Role of Students' Identity Development in Higher Education in Computing*. Ph.D. Dissertation. Uppsala University. Retrieved Mar. 18, 2025 from.
- [52] Carrie Anne Philbin. 2023. Exploring the Potential of Artificial Intelligence Program Generators in Computer Programming Education for Students. *ACM Inroads*, 14, 3, (Sept. 2023), 30–38. Retrieved Oct. 5, 2023 from.
- [53] Leo Porter and Daniel Zingaro. 2023. *Learn AI-Assisted Python Programming With GitHub Copilot and ChatGPT*. Manning, (Nov. 2023). isbn: 978-1-63343-778-4.

- [54] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 2024), 1–31. Retrieved Aug. 29, 2024 from.
- [55] James Prather et al. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, (Dec. 2023), 108–159. isbn: 979-8-4007-0405-5. Retrieved Sept. 3, 2024 from.
- [56] Gregory N. Price and Angelino CG Viceisza. 2023. What Can Historically Black Colleges and Universities Teach about Improving Higher Education Outcomes for Black Students? *Journal of Economic Perspectives*, 37, 3, 213–232. Retrieved July 15, 2025 from.
- [57] Rémi Rampin and Vicky Rampin. 2021. Taguette: open-source qualitative data analysis. *Journal of Open Source Software*, 6, 68, 3522.
- [58] Yolanda A. Rankin and Jakita O. Thomas. 2020. The Intersectional Experiences of Black Women in Computing. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, (Feb. 2020), 199–205. isbn: 978-1-4503-6793-6. Retrieved Mar. 17, 2025 from.
- [59] Yolanda A. Rankin, Jakita O. Thomas, and Sheena Erete. 2021. Black Women Speak: Examining Power, Privilege, and Identity in CS Education. *ACM Transactions on Computing Education*, 21, 4, (Oct. 2021), 26:1–26:31. Retrieved Sept. 27, 2023 from.
- [60] Lauren Rich, Heather Perry, and Mark Guzdial. 2004. A CS1 course designed to address interests of women. *ACM SIGCSE Bulletin*, 36, 1, (Mar. 2004), 190–194. Retrieved Oct. 3, 2023 from.
- [61] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13, 2, (June 2003), 137–172. Retrieved Sept. 27, 2023 from.
- [62] Sarah L. Rodriguez and Kathleen Lehman. 2017. Developing the next generation of diverse computer scientists: the need for enhanced, intersectional computing identity theory. *Computer Science Education*, 27, 3-4, (Oct. 2017), 229–247. Retrieved Mar. 17, 2025 from.
- [63] Herbert J. Rubin and Irene S. Rubin. 2011. *Qualitative Interviewing: The Art of Hearing Data*. sage. Retrieved Aug. 28, 2024 from.

- [64] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. en. *Empirical Software Engineering*, 14, 2, (Apr. 2009), 131–164. Publisher: Springer Science and Business Media LLC. doi:10.1007/s10664-008-9102-8.
- [65] Caitlin Sadowski and Thomas Zimmermann. 2019. *Rethinking Productivity in Software Engineering*. Springer Nature. Retrieved Sept. 9, 2024 from.
- [66] Johnny Saldaña. 2021. The coding manual for qualitative researchers. Retrieved July 25, 2024 from.
- [67] 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 (ICER '22)*. Vol. 1. Association for Computing Machinery, New York, NY, USA, (Aug. 2022), 27–43. isbn: 978-1-4503-9194-8. Retrieved June 21, 2023 from.
- [68] Beverly Daniel Tatum. 2017. *Why Are All the Black Kids Sitting Together in the Cafeteria?: And Other Conversations about Race*. Basic books. Retrieved July 16, 2025 from.
- [69] Vincent Tinto. 2012. *Leaving College: Rethinking the Causes and Cures of Student Attrition*. University of Chicago press. Retrieved July 16, 2025 from.
- [70] U.S. Dept. of Commerce, National Telecommunications and Information Administration. 1999. Falling through the net: Defining the digital divide. Retrieved Feb. 27, 2025 from.
- [71] Etienne Wenger. 1998. *Communities of Practice: Learning, Meaning, and Identity. Learning in Doing*. Cambridge University Press, Cambridge, U.K., New York, N.Y. isbn: 978-0-521-43017-3 978-0-521-66363-2.

Chapter 6

Beyond Efficiency: A Critical Political Economy Perspective on AI Code Generators in Computing Education

Chapter Preamble

This chapter completes Aim 2 by offering a sociological critique of code generating AI and its implications for computing education in the capitalist economy of the United States. Drawing on sociological theories of Marx and Weber, we analyzed interview data from the studies described in Chapters 2 to 5 related to the area of Critical Political Economy (CPE). This sheds light on the capitalist biases that novices will bring into their interactions with code-generating AI in their education and later careers. After considering the social hazards our data signaled about adopting AI in CS education, we suggest mitigating educational interventions, concluding Aim 2 of this

Authorship, Reuse, and Licensing Statement This chapter was coauthored with Claire Wood  and Ellis Barnes , who consented to the inclusion of this chapter in this dissertation. The study explores a dataset that overlaps with Chapter 3, but Chapters 3 and 6 explore different research questions, and this chapter analyzes additional data. For additional information, see the Authorship, Reuse, and Licensing Statement in Chapter 3. All coauthors across both chapters (Sara L. Riggs , Claire Wood , and Ellis Barnes ) provided explicit consent for the inclusion of this chapter in this dissertation. The content from this chapter may appear in a future ACM publication. The license for that paper applies to this chapter. The fallback license is listed on the copyright page for this dissertation. ACM allows complete reuse of published papers in doctoral dissertations per the following section of the *ACM Author Rights* policy, retrieved on March 16, 2026: "Authors can include partial or complete papers of their own in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included."

dissertation.

Abstract

Generative AI tools such as GitHub Copilot are reshaping how novice programmers learn to code, yet their adoption raises urgent questions about the entanglement of technology, education, and capitalist values. While computing education research has largely evaluated AI code generators in terms of performance, cheating, and pedagogy, less attention has been paid to how these tools reproduce broader political-economic ideals. Drawing on Critical Political Economy (CPE), this study examines how capitalist values manifest in students' experiences with code-generating AI and the consequences of those values. We conducted interviews with 27 novice programmers at two U.S. universities who completed programming tasks under time pressure, with and without GitHub Copilot. We identified three interrelated dynamics: (1) rationalization and automation bias, where students privileged efficiency and accuracy by deferring to AI, often at the expense of autonomy, (2) individualism, where students assumed full responsibility for regulating, guiding, and owning AI-produced code, leading to guilt and self-blame, and (3) alienation, where reliance on Copilot estranged students from the labor of programming and from collaborative social relationships. These findings extend sociological critiques of capitalism into CS education, highlighting how students are positioned simultaneously as consumers and producers of commodified intellectual labor. We argue that CS education must cultivate AI literacy that resists defaulting to capitalist focus on productivity and efficiency, instead encouraging critical reflection on how to use AI to promote human welfare within computing. By bridging sociology with computing education, this study offers a framework for understanding the socio-political hazards of AI the CS educational context.

6.1 Introduction

Advancements in generative Artificial Intelligence (AI) have enabled a booming marketplace of Artificial Intelligence-driven Development Environments (AIDEs) that help to produce and maintain computer code in revolutionary ways. Cutting-edge AIDEs come with advanced user interfaces for making use of generative AI in the coding process directly within an Integrated Development Environment (IDE), where a developer writes their code. In the early 2020s, products such as GitHub Copilot and Tabnine began offering inline code completions powered by AI [22]. Even popular generalized chatbots, such as ChatGPT, can act as AIDEs, writing near-perfect code in numerous languages, planning complex software projects, and working with existing code from human developers. Following research about the benefits of chat-style interactions for coding with AI, many of the specialized AIDEs began including the chat paradigm within their feature sets [69]. In 2025, GitHub Copilot and other AIDEs like it offer not only code suggestions and chatting, but also interactive editing, action taking, error detection, and AI-powered codebase search functions to name a few.

As the capabilities of AIDEs progress, they offer increasingly compelling propositions to individuals and businesses seeking to enhance their software development endeavors. In market-driven (i.e. capitalist) economies, the proposition posed is increased profitability, a concept which transfers awkwardly and inadequately into CS education. Whereas profit-seeking prioritizes producing code quickly, programming endeavors in CS education are designed to help students learn foundational skills and concepts [2, 72]. These two mindsets are at odds, but the conflict is far older than modern AI. To attain the capital represented by academic credentials [10], CS students must produce many assignments and assessments consisting of code and code-related media under time pressure ranging from minutes to years. Thus, the structure of curriculum is itself a representation of the market-driven values in which the U.S. education system is steeped. AIDEs complicate this environment by doubling down on the capitalist obsession with productivity and likening the goals of a profit-driven entity to that of every individual themselves. Whereas a sole proprietor producing code for the software marketplace might find such tools as Copilot, be-

ing designed with productivity in mind, as being well aligned with their needs and intentions, the typical CS student might not. Worse yet, the emphasis of the educational system on productivity of educational deliverables means that CS students are being invited by AI vendors to exploit the productivity-enhancing effects of AIDEs rather than exploring their potential learning benefits.

Discussion about the comparative risks and benefits of AIDEs for CS students' learning is lively. Early work frightened the CS education community by demonstrating that AIDEs were rivaling the abilities of CS students on educational deliverables [16, 17]. Despite fears students would soon begin using AI to cheat on their CS assignments, scholars have also considered that AI can help students to better understand coding concepts, remove barriers to succeeding in computer science, and unlock new teaching capacities for busy instructors [5, 60].

Although AIDEs offer legitimate enhancements to traditional modes of learning, teaching, and practicing CS, their current posture in the economic landscape makes them uniquely threatening to CS students learning to code at the start of this new era. Traditional sociological critiques of the CS education system have focused extensively on how it reproduces power structures over time. These draw heavily on the work of Bourdieu [8, 10, 9] and suggest that the omnipresent challenge of diversifying participation in CS owes its roots to nuanced systems of capital [36]. Although such approaches promote resisting unjust power structures [14, 30], they are limited in that they avoid critiquing capitalist economic values. Properly investigating the implications of AI for CS education requires scrutinizing its entanglement with market-driven economics. To this end, the goal of this article is to explore and critique capitalist values that code-generating AI reinforces within CS education.

While sociological theory has been widely applied to study labor, technology, and education, it has rarely been integrated fully into engineering and CS education research. Critical Political Economy (CPE) frameworks offer concepts such as alienation, rationalization, and commodification that illuminate how capitalist logics shape both labor and learning [23, 50, 85]. Yet, in CS education, sociological perspectives are most often limited to individual-level analyses of cultural or social capital, typically in service of diversity and participation goals [10, 36]. What is missing is sustained engagement with theories that interrogate how institutions like education mirror and

reproduce capitalist values, and what consequences follow from these arrangements.

Critiquing capitalism is not new to sociology; it has long been central to the discipline and continues to be a vital point of engagement. From Marx's analysis of alienation and Weber's theory of rationalization, sociologists have used such frameworks to examine how capitalist systems shape our society. More recent work extends these logics through credentialism, bureaucratic organization, and the commodification of knowledge [21, 23]. Similarly, scholarship on technology has demonstrated how machines and digital systems function not as neutral tools but as vehicles for capitalist control, efficiency, and deskilling [11, 65]. Our contribution builds on this tradition by applying these sociological critiques to computer science education and the adoption of AI code generators. In doing so, we show how capitalist values embedded in technology and education intersect in CS classrooms, shaping novice programmers' experiences and raising new questions about labor, learning, and autonomy in the age of AI. In the educational context, we reveal how students are positioned not only as consumers of knowledge, but also as producers of exchange-value, where their intellectual labor is commodified.

This paper bridges the gap between sociology and CS education by bringing Marxist and Weberian frameworks into dialogue with empirical CS education research. This study contributes a novel critical framing of code-generating AI in U.S. CS education using interviews from 27 novice programmers at two universities who programmed under time pressure with and without the GitHub Copilot AIDE. Through a systematic qualitative analysis process, we mapped out an interconnected set of capitalist values present in participants' reflections on programming with Copilot compared to programming alone. By doing so, we move beyond describing student outcomes to explaining how capitalist economic values structure novice programmers' encounters with AI code generators.

Research Question How can novice programmers' thoughts and feelings about using GitHub Copilot to code be explained by the entanglement of the United States computer science education system with capitalist economic values?

6.2 Theoretical Framework

6.2.1 Critical Political Economy

Critical Political Economy (CPE) is a theoretical approach that interrogates how money, politics, and power shape and control resources according to dominant ideologies and power structures [23]. Originating from classical political economy in the works of scholars like Adam Smith, David Ricardo, and Karl Marx, CPE examines how capitalism organizes social relations, shapes power dynamics, and influences technological interactions and labor processes [50, 62, 76]. More specifically, it is rooted in Marxist analyses that critique capitalism not just as an economic structure but as social totality. Marx focuses on the value of labor, how people can feel disconnected or "alienated" from their work, and how capitalists gain profit from the labor of workers [50, 49]. These ideas help explain how capitalist systems create and maintain patterns of wealth and power.

Marx's contributions remain central to CPE, especially his exploration of commodification and alienation. Commodification is the abstraction of a product's unique value for people (i.e. "use-value") to a quantifiable "exchange-value" suitable for commerce [49]. Alienation refers to the estrangement workers experience from their labor, the final product of their labor, their fellow workers, and ultimately themselves through the extraction of their labor as a resource by those who own the means of production [50]. Specifically, Marx says:

"The alienation of the worker in his product means not only that his labor becomes an object, an external existence, but that it exists outside him, independently, as something alien to him, and that it becomes a power on its own confronting him; it means that the life which he has conferred on the object confronts him as something hostile and alien." [50]

An example of this concept is found in Braverman's [11] analysis of factory labor, where increased mechanization and managerial control resulted in workers losing autonomy, feeling reduced to mere operators of machines rather than skilled artisans. This condition intensifies as workers become subordinate to technology, increasingly operating as mere appendages of machines rather than as autonomous creators [49].

Marx's theoretical framework identifies four distinct forms of alienation—alienation from products, labor processes, self, and social relations—each created because of capitalism [50]. More recent scholarship [54] argues that alienation should not be viewed merely as individual feelings of dissatisfaction. Instead, alienation is a systemic and objective condition arising directly from the division of labor within capitalism. In this framework, alienation occurs when workers are separated from the products of their labor, which are appropriated, commodified, and ultimately used to limit their autonomy. Additionally, contemporary forms of alienation are not confined to the factory floor but extend into cognitive and affective domains, where creativity and emotional labor are commodified [54]. This process creates a paradox where society appears to become more connected and interdependent, yet individuals feel increasingly isolated and powerless due to the control and impersonal nature of capitalism [52, 54].

Additionally, in Marx's critique of capitalism there is a distinction between use-value and exchange-value. Use-value refers to the practical utility of a commodity—the capacity of an object or service to satisfy a human need. Exchange-value, by contrast, reflects the commodity's worth in the marketplace, expressed in its relation to other commodities or monetary terms [49, 79]. While use-value anchors commodities in concrete human needs, exchange-value abstracts them into quantifiable units of comparison, enabling their circulation as capital. Commodification occurs when the unique, qualitative dimensions of use are subordinated to the quantitative logic of exchange, such that the social relation between producers and consumers appears as a relation between things [49, 71]. This tension between use and exchange is at the heart of CPE analyses, revealing how capitalist systems prioritize profitability and accumulation over human need, and how institutions—including education—reproduce these dynamics by treating knowledge and skills as commodities whose exchange-value outweighs their use-value.

6.2.2 Rationalization

Alongside Marx, Max Weber's theory of rationalization provides additional explanatory power when considering the impact of Artificial Intelligence within computer science education. In *The*

Protestant Ethic and the Spirit of Capitalism, Weber explains the shift between traditional ways of life and the industrialized modern world [85]. Analyzing the content of discourse on labor prior to the Industrial Revolution, he describes how Protestant workers equated economic success within divine favor, subsequently embedding the values of efficiency and wealth accumulation within work. As a result, over time, the traditional and religious customs that informed labor gave way to reason, logic, and science, with work being valued for its economic benefits rather than for tradition or passion. For Weber, this shift, what he calls rationalization, reflects a preference for efficiency that now defines modern society [85].

This leads to what is known as *instrumental rationality*, the idea that human activity should primarily serve practical, goal-oriented ends [6]. Bureaucracy is one institutional form of this mindset, in which organizations are divided into specialized units to maximize efficiency and productivity. More than a structural description, Weber viewed bureaucracy critically: its emphasis on standardized procedures ensures smooth operation for economic gain but also subjects people to domination, limiting autonomy and stifling creativity [85].

Building on this, bureaucracy can be understood as a mechanism of control rooted in knowledge. By standardizing labor, it assigns individuals to predefined roles and strict procedures, aiming to minimize variation in outcomes by privileging the most efficient means of generating economic gain [32]. This makes instrumental rationality the dominant framework for all decision-making, replacing moral and communal bonds with competitive, individualistic structures. Weber argues that this is an irrational system of governance where the pursuit of efficiency leads to a restrictive and alienating social order [19, 32, 85].

Weber's theory of rationalization rests on an interpretive approach that examines how individuals make sense of their actions. While this may appear psychological in nature, it is fundamentally sociological. The emphasis is not on isolated mental states, but on how orientations toward action reflect broader social structures, values, and norms [6, 13, 68]. In this sense, analyzing individual meaning-making reveals how cultural expectations and institutional arrangements shape thought and behavior. This sociological perspective provides a useful foundation for our study, as it highlights how students' experiences with AI are embedded in—and shaped by—capitalist

systems.

6.2.3 Technology

Marx saw technology not as inherently liberating or oppressive, but as a force shaped by the social and economic context in which it is developed and employed. In capitalist societies, he argued that technological advancements are typically used not to reduce human suffering or labor but to increase profits and control over workers. Machines and tools become instruments of alienation when they are used to deskill labor, speed up production, and maximize surplus value for the capitalist class [50, 52].

Braverman identifies two fundamentally different perspectives on technology. The first is the engineering perspective which understands technology primarily through its internal functions, defining machines by their technical characteristics. The second is the social perspective, which emphasizes technology's relationship to human beings and defines machines according to their connection to human labor [11]. Within capitalism, the social perspective dominates: technologies are designed and implemented to extract value and maintain control over workers and their products.

This tendency reflects what Ritzer terms *McDonaldization*, a contemporary iteration of Weber's theory of rationalization. Within McDonaldization, modern institutions (e.g. travel, retail, education, etc.) operate according to the same principles as the fast-food industry: efficiency, calculability, predictability, and control through non-human technology [65]. These principles standardize processes to deliver predictable outcomes for profit maximization. Control through non-human technology, the final tenet of McDonaldization, is central to this logic.

With increasing digitalization, more tasks are automated under the premise that machines reduce variability and error [66]. Digital kiosks replace cashiers, turning customers into unpaid laborers for corporations [63]. In manufacturing, the industrial internet embeds sensors, trackers, and chips into machinery so that production components communicate autonomously, bypassing the need for direct worker or managerial oversight [77]. Modern information technologies are

celebrated as accelerants of innovation, competition, and economic gain for tech companies [40]. Human laborers become valued more for their ability to operate interfaces and follow protocols than for their capacity to generate original solutions [29, 37].

The consequences of widespread technological adoption extend beyond reducing human autonomy; they also transform how laborers relate to their work. *Automation bias* describes the tendency to over-rely on automated cues as substitutes for the cognitive effort of seeking and processing information [51]. This bias often manifests as a preference for suggestions generated by technology, even if those suggestions conflict with other available information. Research shows that excessive dependence on automation—commonly used to manage repetitive and monotonous tasks—can foster complacency, degrade skills, and erode situational awareness [55]. The rise of AI in the modern world is not an exception to this trend but an extension of it. The use of AI in coding represents a form of attentional integration within digital labor that may discourage programmers from relying on their own skills in favor of the convenience AI provides.

For computing educators, adopting a critical lens is essential for understanding how the tools they create operate within broader political-economic systems. Ultimately, technology under capitalism is less of a neutral instrument than a conduit of profit and control. From Marx's analysis of machines as instruments of alienation to Braverman's social perspective on technology, technological design has long served capital rather than labor. Contemporary forms—i.e., McDonaldization—extend this logic and illustrate how efficiency and predictability are achieved by reducing worker autonomy and creativity. AI and automation bias mark the newest turn in this trajectory, drawing workers into patterns of dependence that subtly reshape how they engage with their own labor.

6.2.4 Education

Marx imagined capital as a tangible means of production, but modern scholars argue that our current economy is driven by intangible capital [7, 70, 25, 70]. Intangible capital refers to non-physical assets that generate value, such as knowledge, intellectual property, algorithms, etc.

Unlike tangible capital—factories, machines, or raw materials—intangible capital assets are not material objects but forms of knowledge and innovation that can be leveraged for economic gain. In contemporary capitalism, these assets underpin value creation, shifting the focus from physical production towards cognitive, informational, and cultural resources.

Education under capitalism often mirrors broader economic structures, reinforcing the systemic inequalities central to capitalist production. Contemporary education is increasingly shaped by intangible capital—such as knowledge, skills, and credentials. These intangible assets are treated as forms of capital investment, expected to yield future economic returns for both individuals and institutions. This framing repositions students not just as learners, but as future capital holders and entrepreneurs of the self, whose value is tied to their ability to accumulate and mobilize intangible capital in competitive markets.

Pierre Bourdieu's concept of cultural capital further clarifies this dynamic. Cultural capital refers to non-economic resources such as educational qualifications, knowledge, and social behaviors that confer status and advantage in society. In educational contexts, cultural capital is often aligned with intangible capital, influencing students' ability to successfully navigate and exploit educational systems and labor markets [10]. He argues that educational institutions serve to legitimize and reproduce existing social hierarchies by rewarding students who possess certain forms of cultural capital, thereby perpetuating inequalities and reinforcing capitalist class structures [10].

Henry A. Giroux is a central figure in critical pedagogy and one of the most influential voices applying Critical Political Economy to the analysis of education. His work challenges how neoliberalism transforms educational institutions from spaces designed to foster democratic engagement into systems oriented around market logic and private gain. In *Neoliberalism's War on Higher Education* [21], Giroux explains that universities are increasingly structured like corporations: students are treated as consumers, faculty as managed labor, and knowledge as a commodity for sale. He critiques the erosion of public funding and the growing influence of private capital, which results in universities prioritizing job training and economic output over critical inquiry and civic responsibility.

In the capitalist education system, students are simultaneously framed as both consumers and producers of value. The educational system, particularly in neoliberal contexts, is structured to reproduce labor power attuned to the needs of capital. This includes the emphasis on STEM over the humanities, the rise of credentialism, and the privatization of educational services [12, 21].

Structurally, education reflects broader capitalist dynamics through its bureaucratic organization. For nearly two centuries, bureaucracy has shaped educational institutions, characterized by extensive administration [24, 28]. Such bureaucracy imposes significant structuration within academia, with disciplinary divisions controlling the types of knowledge produced and promoted by different departments [1]. Consequently, this fosters intense competition over specialized forms of knowledge, positioning universities as hyper-specialized bureaucratic entities deeply involved in knowledge production and mediation [15, 78].

Similarly, standardized course loads for students perpetuate intellectualism as a profit-making device as the value in higher education shifts from the quality of knowledge one can obtain to how effectively the university can produce individuals ready for the workforce [15, 24]. One could argue that vocational qualifications secure a level of quality in knowledge generated and mediated to students, but even these represent bureaucratic standards that are in place to ensure homogenized operations.

These perspectives illustrate how education functions as a critical site in the political economy of capitalism. The rise of intangible capital has reframed knowledge, skills, and credentials as investments, while cultural capital shapes who benefits from these processes and who is excluded. Under neoliberalism, universities increasingly mirror corporate logics, commodifying knowledge and redefining students as both consumers and producers of value. Bureaucratic structures further entrench these dynamics, standardizing knowledge and aligning educational outputs with the demands of capital. In this way, education not only reflects broader economic transformations but actively reproduces them, reinforcing inequalities while legitimizing the capitalist order.

6.3 Related Work

6.3.1 Social Theory in CS Education

The use of sociological theory in CS education research is mostly limited to promoting diversity by considering capital at the individual level. A popular foundation is Bourdieu's theory of cultural reproduction, which is built on his understandings of capital, its embodiment in individuals (habitus), and its location of use (field) [8, 10, 9]. In addition to economic capital (i.e. capital and property), Bourdieu emphasizes cultural and social capital, where credentials and relationships are also assets. CS education research focuses on interventions that promote participation from underrepresented students by helping them attain and recognize cultural and social capital [30, 36, 84]. Another branch of research attempts to resist hegemony using pedagogies that encourage students to reshape the dominant culture within their education [14, 61, 73, 80]. However, there is limited work that critiques the CS education system using sociological theories about capitalist economies [41, 42, 47].

6.3.2 AI in CS Education

Artificial Intelligence-driven Development Environments (AIDEs; i.e. AI code generators) such as ChatGPT and GitHub Copilot offer new ways to assist computer science students with programming and related tasks. In the past, CS students sought help from social resources like peers, instructors, online communication and nonsocial resources like courses, books, and online information [27]. AI is an asset for CS education but also introduces ethical and other challenges such as overreliance and influences on students' coding habits [5]. CS instructors are learning to cope with AI [60], but are not fully aligned with students regarding appropriate policies [91]. Tools such as CodeHelp [44] can provide AI assistance to CS students without generating code, but code-generating AI nevertheless remains widely available to students.

6.3.3 Performance and Learning Effects

Though AIDEs improve novices' programming performance, their benefits for learning outcomes are not yet clear. Early work was inconclusive about the performance benefits of AIDEs for programmers in general [83, 88], but more recent evidence has solidified that AIDEs improve novices' programming performance [18, 38]. Although efficiency potentially allows students to cover material more quickly, what matters more in an educational context is whether students are meeting learning objectives. AI-inclusive CS pedagogies have begun to win the appreciation of students and instructors alike [46, 82, 91]. Unfortunately, AIDEs have not been found to significantly impact learning outcomes overall [35, 38, 89]

AIDEs only seem to improve learning when used in certain ways, highlighting the general inconsistency of AI's impacts. The research is mixed about how CS students choose to use AI when they are allowed; some use it to outsource critical thinking, while others use it as a supplement [2, 39, 48, 59, 58]. A study of 32 CS students found that using AI more for critical thinking tasks (code generation and debugging) was significantly correlated to worse learning outcomes [35]. This aligns with the findings of [34] that friction between AI outputs and a CS student's preconceived ideas is a catalyst for meaningful engagement with the subject matter. To reach a point of friction, the student needs to have put forth sufficient thought about a programming task to have a basis for comparison. In this vein, Kazemitabaar found that young CS learners (n=33) who used AI for some but not all work showed a non-significant trend toward better learning outcomes compared to full insourcers or outsourcers [39]. Overall, whether AI is beneficial for learning computer science seems to be highly variable according to different circumstances and choices.

6.3.4 Social Effects

By mimicking human assistance, AIDEs present a threat to social learning in CS. Existing approaches, such as student pair programming, are beneficial to students' well-being and performance [74]. AIDEs are an avenue for programmers to avoid social help-seeking [27, 83]. Within

pseudo-social AI interactions, students can attain their primary objectives for seeking help without the undesirable properties of socializing with humans. For example, CS students find that AI offers less judgement compared to peers or instructors [46]. Furthermore, programmers often see AIDEs as CS experts [20, 27], meaning they expect superior assistance from AI as compared to, for example, a peer. Once programmers begin to go to AI for help, they tend to stray away from other resources [89]. This transition reflects the theory of rationalization; the obsession of students to achieve academic performance metrics causes them to devalue social connections within the context of learning to code.

6.3.5 Self-Efficacy Effects

Student self-efficacy, which is a strong positive predictor of academic performance [26], can potentially be boosted by AIDE use. One study (n=45) has shown that using AI in a CS course provides significant benefits to self-efficacy and motivation [90]. However, in another study of CS students, earlier and more frequent AI use was conversely associated with low self-efficacy and course grades [48]. In empirical studies, programmers using AI tend to blame themselves for negative outcomes rather than blaming the AI [27, 53, 86] yet do not always take credit for AI performance gains [18]. A possible reconciliation of these findings is that struggling students with low baseline self-efficacy seek AI assistance early and often, leading them to gain a false sense of competence [58]. When the programmer's true competence is more transparent, as in [18], self-efficacy is less likely to improve as a result of AI assistance.

6.3.6 Autonomy Effects

AIDEs reduce CS students' autonomy. In empirical studies, programmers report desiring to maintain control over AIDEs while programming [43, 83], but this is not often borne out in reality. For example, several studies have found that programmers heavily modify their work processes in response to the presence of AI [3, 43, 69]. AIDEs can also be intrusive and can cause novices to lose track of their progress within a programming task [31, 87]. As mentioned earlier, the re-

sulting "illusion of progress" can cause a false sense of programming competence [58]. Loss of autonomy is especially likely when a CS student lacks knowledge on the subject at hand. A study in a CS course (n=56) had students take assessments with ChatGPT prior to learning the relevant course material. Though students were capable of achieving correct answers, their incorrect answers indicated they were easily misled due to lacking the necessary knowledge to identify bad advice [75]. In an economy that is already heavily bureaucratized and siloed, losing autonomy over programming further alienates programmers from the larger societal benefits of their everyday work.

6.3.7 Inequality Effects

AIDEs are likely to worsen existing inequality in CS education. Several studies support that AI is more useful to stronger CS students than weaker ones [38, 48, 58]. AI access is another source of inequality. As of 2024, internet resources still dominated AI for CS students seeking help, but among AIDEs, ChatGPT dominated GitHub Copilot, though the latter is designed more specifically for programming [27]. That is, students are using the most accessible resources, not necessarily the most effective ones. In the past, disparities in access to the internet reinforced existing economic inequalities [81]. Since AI-inclusive pedagogy is new, it will also likely cause widened educational inequality between more and less resourced institutions according to who can afford to invest in it. Many studies have shown that effective shepherding of AIDEs is key to their effectiveness [3, 20, 31, 33, 39, 43, 53, 59, 69, 83, 86]. The long term effects of these factors are not yet clear, but evidence for inequality is already surfacing. At least two studies have found that student use of AI code generators is bimodally distributed among students who are familiar with them [27, 48]. Unless CS education deliberately combats these effects, what is likely to follow is a "Matthew Effect," whereby students' existing advantages reinforce themselves [64].

6.4 Method

6.4.1 Participants & Context

The data were collected alongside a controlled laboratory experiment comparing various subjective and objective measures between solo and AI-assisted programming conditions. Two cohorts of participants were recruited at two public research universities in the same U.S. state: a Predominantly White Institution (PWI) and a Historically Black University (HBCU). Participants were recruited by convenience, taking volunteers from introductory programming courses. Ethical approval was obtained for both studies, and all participants provided informed consent and were paid for their time. Interested volunteers participated in the study outside of class, and no volunteers were turned away.

PWI Cohort

The PWI cohort included 17 students from an introductory CS1 course taught in Python who identified mostly as female (11 female; 5 male; 1 unknown) and Asian (8 Asian; 3 Asian & White; 4 White; 1 Latinx & Black; 1 Latinx & White). A recruitment announcement was made in two of three sections of the CS1 course to a total of roughly 500 students in Spring 2023, when data were collected.

HBCU Cohort

The HBCU cohort included a gender-balanced sample (5 female; 5 male) of only Black students from a CS2 course taught in Java that immediately followed the Python CS1 course. Based on comparing the syllabi for the CS1 courses at both universities, recruiting from the CS2 course ensured the most similarity between the ability levels of the two cohorts. The recruitment announcement was made to all five sections of the CS2 course to a total of roughly 100 students in the Spring of 2024, when data were collected.

Experimental Context

The qualitative data in the present study comes from interviews conducted following participants' completion of a controlled programming experiment. Note that the focus of the present study is on what participants' reflections on their programming experiences say about AI's role in reinforcing capitalist values in CS education. As such, the details of the programming experiment are not the major contribution here, but we provide them for necessary context.

Programming Tasks

Tasks were selected from the HumanEval dataset, which is used to evaluate LLMs' coding abilities. It is intentionally published in a compressed format to mitigate its inclusion in training data. Despite efforts to exclude it during training, the open-source nature of HumanEval means GitHub Copilot may unintentionally be trained on some of its tasks. All tasks provide a prompt, typically with example inputs and outputs, and expect a Python function to return an output based on input parameters. Each task comes with a set of test cases to check a solution's correctness. An example task (id=104) includes the following prompt: "Given a list of positive integers x, return a sorted list of all elements that hasn't any even digit". To avoid this article contributing to benchmark pollution, we will not provide any solutions to HumanEval tasks.

Nine teaching staff from the CS1 course at the PWI helped to examine and select tasks of tractable but varied difficulty to use in the study from the HumanEval dataset. Using the consensus of the multiple raters, four sequences of tasks from HumanEval were constructed to have easier tasks toward the front, harder tasks in the middle, and tasks requiring special knowledge like math formulas at the end. The goal was for all sequences to have similar difficulty profiles to one another, without participants getting stuck early. Participants were required to work on tasks one-by-one so that time-management did not confound the results. To move to the next task in each sequence, participants had to pass all of the current task's unit tests, which are provided within HumanEval. The task sequences, by task id, were: (24, 3, 0, 12, 26, 37, 51, 49), (53, 52, 18, 69, 25, 70, 7, 2), (23, 74, 14, 9, 6, 161, 19, 22), and (35, 67, 72, 5, 13, 38, 20, 45).

AI is poised to perform better on HumanEval tasks than real-world programming tasks, but the main reason for this is their simplicity; this will be an issue for any task that is simple enough for a novice to attempt. Anecdotally, both teaching staff and participants agreed that the selected HumanEval tasks closely mimicked the types of tasks that would be typical for programming assignments and timed assessments in a Python CS1 course.

Programming Setup

Participants programmed in Python on Apple iMac computers using Visual Studio (VS) Code. A custom VS Code extension provided a button to control the task progression. A first press began the eight task sequence and started a 20-minute timer. This placed all HumanEval instructions for the first task in a PDF file on the right side of a split screen and opened a Python file on the left. The Python file contained an empty function to implement, stripped of all comments and instructions. This prevented Copilot from ingesting the task prompt unless the participant explicitly introduced it from the PDF via comments or chat messages. A Python testing and running interface was provided by the standard Python extension for VS Code, and a terminal was available for running and testing, too. The unit tests came directly from the HumanEval dataset and could be seen but not changed by participants. After implementing, testing, and running the code ad libitum, the study control button would allow them to advance to the next task if their implementation passed all tests. Otherwise, they could not advance. Upon the expiration of the timer, the code was automatically submitted and evaluated as-is. While the first cohort was not allowed to use internet resources at all, the second cohort was allowed to do so in both AI and solo conditions on the advice of our peers in the CS education community. This tradeoff sacrificed exact methodological parity for increased ecological validity.

The GitHub Copilot VS Code extension was available when participants worked with AI, while the Python extension provided standard symbolic autocomplete for solo programming. Copilot is a popular mainstream code-generating AI assistant that is free for students and teachers. Because GitHub Copilot is constantly being evolved and updated, the version used by the second cohort was slightly different than for the first. Both cohorts had inline suggestions for code within

their editor and the ability to invoke a panel of multiple completion options following their cursor position. However, the HBCU cohort also had chat features and context direction tools available. Again, it is important to note that Copilot could not read the task prompts from the PDF instruction files.

Programming Procedure

Participants provided informed consent before training extensively on GitHub Copilot and the experimental setup. They then programmed for four 20-minute bouts corresponding to each of the four task sequences, using GitHub Copilot for two bouts and working alone for two more. The ordering of the task sequences and the assignment of the AI treatments was counterbalanced across participants, with the 10 HBCU participants' task lineups mirroring 10 of those from the PWI. Participants were incentivized by two \$10 bonuses offered within each cohort (four total) to complete as many tasks as possible, using AI as much or little as needed. They were scored based on how many total tasks for which they passed all unit tests over all programming bouts.

Interview Procedure

Semi-structured interviews were conducted using the five-phase approach described in Chapter 7 of [4]. This format eases participants into and out of the main topics of interest and avoids ambiguity and bias by asking open-ended questions and following up on markers of key experiences and sentiments. One author performed all 27 interviews. The goal of the interview was to explore the subjective experience of programming with Copilot under time pressure and its impacts on participants' thoughts and feelings (see Chapter 3 and appendix A). Member checking [45] was performed frequently throughout all interviews to facilitate valid interpretations during analysis. However, it was not feasible to perform post-hoc member checking with participants for the findings presented here. Interviews were audio recorded and transcribed using automated transcription, with revisions made manually by a trained researcher.

6.4.2 Data Analysis

During an initial review of the interview transcripts from the 17 participants at the PWI, one of the authors noticed a pattern that many participants' comments alluded to underlying capitalist values. This inspired us to undertake a deductive qualitative investigation focused specifically on classifying, explaining, and critiquing these patterns. To do so, three trained qualitative researchers analyzed the data in four phases, beginning with a deductive approach in phases one and two and moving to an inductive approach for phases three and four.

Phase 1: Quote Selection

In the first phase, one researcher independently reviewed each interview transcript, identifying quotes related to the research questions. The second researcher reviewed each transcript and could add new quotes, but was not permitted to remove any previously selected quotes. This ensured a comprehensive selection of quotes that could later be refined for relevance. Researchers followed the broad rule: "Select any quotes possibly reflecting or resisting capitalist economic values." Including duplicates, this phase yielded 190 quotes from the HBCU and 611 from the PWI for a total of 801.

Phase 2: Quote Pruning

In the second phase, two researchers independently reviewed all quotes from Phase 1, flagging any that seemed irrelevant to the research questions. We used a crossover strategy, so each of the three authors covered two thirds of the data and overlapped by one third with each of the other authors. The shared instruction was: "Flag any quotes that do not relate to capitalist values or do not reflect an idea or experience of the participant's own (e.g. I don't think X but some people might)." For each flagged quote, the selector and the flagger met to resolve the disagreement and decide whether to keep or discard the quote. This reduced the total dataset to 397 quotes.

Phase 3: Initial Coding

In the third phase, two researchers independently coded each quote from Phase 2 using an inductive approach and the same crossover strategy. While the quotes were identified through a deductive process, coding remained inductive—there were no predetermined limits on which capitalist values might be relevant. The shared guideline was: "Label and describe the relevant capitalist values reflected or resisted by the quote."

Phase 4: Codebook Synthesis and Refinement

In the fourth phase, all three authors met to merge all codebooks and reevaluate any quotes irrelevant to the research question. We reviewed each code, combined similar codes, clarified descriptions, subcategorized codes under others as needed, and eliminated some. Each decision was discussed and approved by at least two of the three researchers in the case of disagreements. Finally, we checked that each code was internally coherent, distinct from others, and grounded in the data. Each theme was supported with representative quotes and revisited in relation to the research question to confirm relevance. The resulting themes are discussed in the next section. In total, the final dataset included 217 quotes, of which roughly 5% are reported below.

6.5 Results

Participants' quotes revealed four main themes reflecting capitalist values in relation to programming with code-generating AI. First, they reflected an *Automation Bias* that encouraged them to use Copilot rather than engage with programming tasks. Second, they promoted *Individualism* by projecting all aspects of interacting with it onto a reflection of themselves. Third, they demonstrated Copilot's *Alienation* of the programmer from the programming process and social opportunities. Fourth they revealed the underlying tendency of *Rationalization* (i.e. prioritizing efficiency and other measurable outcomes) that explained the observed *Automation Bias*.

Table 6.1: Qualitative codebook summary.

Theme	Participants	Description
Automation Bias	25	Using Copilot specifically to avoid engaging with a programming task
Passivity	20	Loss of autonomy in the programming process
Temptation	16	A lasting desire to use Copilot again
Pragmatism	7	Using Copilot because it is more efficient
Individualism	21	Burdening oneself as personally liable for all aspects of using Copilot
Responsibilities	21	Expecting the user to regulate their use of Copilot, use it effectively, and take ownership of the resulting code and its consequences
Self-blaming	9	Assuming Copilot's underperformance is one's own fault
Guilt	6	Feeling bad for using Copilot excessively
Alienation	20	Estranging the programmer from themselves, their programming labor, and others people
Labor Alienation	18	Loss of pleasure in programming and identity and perceived importance in the programming process
Social Alienation	8	Substituting Copilot into otherwise human social roles
Rationalization	19	Prioritizing calculable goals over holistic or subjective ones
Idolizing Technology	15	Assuming that technology is good for society and helpful for most needs
Efficiency	8	Believing faster is generally better
Optimality	5	Desiring not just satisfactory quality but perfection

Note. The four themes and their first-level subthemes are defined. A count of the number of participants out of 27 who shared a quote coded with each code is also reported. The order of themes follows this measure of prevalence in contrast to the narrative layout within the subsections to follow.

6.5.1 Automation Bias

Nearly all participants reported applying an *Automation Bias* while coding with Copilot. Merely using Copilot was not considered here; instead, quotes specifically related to using Copilot as an alternative to manually engaging with a task. This included the main purpose for this bias (*Pragmatism*), its expression as *Passivity*, and its consequence (*Temptation*).

Pragmatism

Pragmatism is the belief that using Copilot is more efficient for coding than not using it. The study design attempted not to enforce this belief, as participants were taught both benefits and drawbacks of Copilot in training and instructed to use it only as much or as little as needed to score as highly as possible. Therefore, their preference to use Copilot over engaging with a task reflected a bias, whether pre-existing or discovered, that using this automation would be better than not. They described how it saves time, makes coding more efficient, and helps get things done because it has the answers they need. For example, participant j24 began by trying to collaborate with Copilot, but realized pasting the entire task description into the chat panel was more successful. Similarly, participant i47 found that even when they did not need Copilot, they were better off with taking its suggestions.

"At first I started asking it to explain because I was trying to, you know, figure it out. But the best strategy was to just copy and paste." (j24)

"I think this was one of the ones where if I had more time, I definitely didn't need Copilot, but... the ease of Copilot definitely made it more like learning to just go with what Copilot generated because it would be correct." (i47)

Participants understood that, like in the study, efficiency would often be a primary goal in their future coding endeavors. For example, participants j11, j16, and j20 said they would use Copilot when they just need to get something done for school, potentially revisiting the work later. Participant j13 emphasized that this approach would be inappropriate for school but would

be acceptable in the workplace, where productivity is the main goal. Generally, all agreed that when productivity or efficiency is a priority, using an AI code generator is obviously preferable to completing a task alone. This likely reflects the reality that they were more productive with Copilot than without but could also be influenced by a general bias to believe automation is superior to manual work.

Passivity

Passivity is the act or mode of deferring to the advice of Copilot over one's own thoughts, i.e. the manifestation of participants' *Automation Biases*. For example, participant i16 said that they sometimes gave Copilot complete control, while i19 explained how it felt to take a passive role with Copilot.

In some cases, "Copilot just solved the problem for me and I didn't do anything" (j16)

The difference isn't huge, but "it's less mental taxation on you so you're more calm"
(i19)

In many cases, passivity was an intentional or *Voluntary* decision to avoid a traditional way to solve a problem and opt for Copilot's advice instead. For participant j4, they chose not to think about a problem when Copilot could do it for them. For j13, they regretfully recalled choosing Copilot over Google search.

"I just asked the AI on this one, and it also gave me the answer and it was correct, so I just moved on... I did not really think too much... I just trusted the AI." (j4)

"Once I started using Copilot, I stopped using Google. So maybe for that [task], I could have consulted Google to see exactly what's the problem and not relying solely on me going back and forth with Copilot." (j13)

On the other hand, Copilot's automatic suggestions meant participants were often forced into an *Involuntary* state of *Passivity*. Compared to the prompt-response model of Copilot's chat or

popular apps like ChatGPT, unprompted suggestions in the editor encourage users to take a back seat. Participant j20 explained the difference.

"With ChatGPT you have to kind of do a little bit of it by yourself before you can input it... but [with Copilot], it's spoon feeding you." (j20)

When Copilot offers its suggestion, any thought process a user has begun is hijacked by the new information. "Your eyes are sort of betraying you" when Copilot's suggestions appear "before you can think" (i47). Users are forced to adapt to the direction Copilot takes them, which can be frustrating and disorienting, as noted by participants i42 and j15.

"I got like, sucked into the paths that didn't work, and then I was stuck there, like I found it difficult to get out of that hole." (i42)

"It was constantly affecting my bias... it was just suggesting things that were not useful... being inline... it changes your bias and just makes you like stumble over it over and over again" (j15)

A takeaway is that the user interface for an AI code generator has a clear impact on the power of *Automation Bias*. When relying on Copilot required deliberate action, such as sending a prompt, *Passivity* was a result of a *Voluntary* choice to prefer Copilot's help. However, the *Involuntary* side of *Passivity* showed that preferring one's own ideas was much more difficult when Copilot polluted their thoughts.

Temptation

Temptation is a durable effect of *Automation Bias*, where users struggle not to use Copilot after knowing or experiencing its practical benefit. Copilot has a way of imprinting on a user such that they become frequently distracted by considering the opportunity cost of coding alone, as in the case of participant j20.

"Even if I decided I was going to ignore Copilot... while going with my own approach, I was still thinking about the possibility of using Copilot, like the opportunity cost" (j20)

Participants also described *Temptation* as the alluring pleasantries associated with using Copilot to avoid work or discomfort. For example, participant j1 felt more secure working with Copilot, and i9 expected Copilot to discourage them from bothering to remember common library functions.

"When I have AI, it feels like more security, like... [whether] it's right or wrong, I could check it or AI can help me get it right." (j1)

Say "I forgot what this function is, but it's okay because I know Copilot will probably be able to help me out as long as I remember at least a part of it." (i9)

The memory of Copilot and its utility is not easily forgotten, leading to an experience participant i13 described as "my heart says yes but my mind says no." That is, even when a user wants to solve a problem themselves, they are drawn to prefer Copilot anyway.

6.5.2 Individualism

Individualism is the spirit that Copilot's users, rather than Copilot or its producers, should be culpable and accountable for their use of it. Participants believed that it was an individual's responsibility to self-regulate their use of AI, control it effectively, and take personal accountability for the code produced. As a result, they felt guilty for using Copilot, and when the interaction was unsuccessful, they blamed themselves.

Responsibilities

The primary sign of *Individualism* in participants' accounts was their identification of moral *Responsibilities* related to using Copilot. First, the responsibility of *Self-Regulation* is to decide when and how much to use AI. Second, the responsibility of *Shepherding* [59] is to effectively understand a problem and coerce AI to be useful. Third, the responsibility of *Ownership* is to be accountable for the code produced as well as an understanding of how that code works.

The responsibility of *Self-Regulation* implies that there can be a wrong time, a "too much" (i19), or a too little associated with using AI. For example, participant i47 said a beginner should

avoid using Copilot, since they would not be prepared to intervene if something were to go wrong. In other cases, participants simply stated that Copilot could have a positive impact on their learning or productivity, but only when used properly, with the proper manner being ill-defined and ambiguous. Participant i11 planned to stay away from AI more after the study, opting to work alone and only use Copilot very sparingly. Their implication is that using Copilot for too much of a project is problematic.

"Trying to do it by myself as long as I can would be better, but if I need a suggestion, one little suggestion, not the whole project, that could be helpful" (i11)

The responsibility of *Shepherding* [59] says the user has a duty to effectively understand a problem and communicate their intentions to Copilot. The sentiment was that Copilot does not owe the user a seamless experience, but rather they must lead the effort. For example, participant i12 explained that identifying their intention was a key step to succeeding with Copilot.

"There were some tasks where I didn't fully understand the question, and if I don't understand the question then I don't know what I want, so Copilot can't know what I want" (i12)

The responsibility of *Ownership* attaches a user to the code they produced with Copilot, such that they must be able to understand its design and answer for its behavior. Participant i5 captures both of these aspects, saying that they must understand the code and act as an authority on the work for future inquiries.

"I want to be the authority on my own work, and I want to understand it, so if... it's only affecting me then I could control kind of how that's going to play out, but if I were to ever have someone ask me about it, or I need to modify something, I'd want to be able to fully understand what I had submitted." (i5)

This quote also suggests that future consequences or retribution motivates why to understand code even when Copilot can produce it correctly. Because one's code can affect others, someone might interrogate a piece of code as a reflection of the user who oversaw its production. The user

also wants to help their future self by investing in an understanding of code now so that it is easier to revisit or maintain in the future.

The responsibilities of *Self-Regulation*, *Shepherding*, and *Ownership* span the full life cycle of Copilot's involvement, capturing the spirit of *Individualism* within programming. A contrasting mindset would be collectivist, meaning other team members, AI vendors, or the AI itself might be expected to share the effort of producing code responsibly. However, it seems that participants believe the individual is ultimately responsible instead.

Guilt and Self-Blaming

The sense of responsibility also led participants to have negative feelings and reflections toward themselves. Even though they were allowed to use Copilot as much as they wanted, they felt *Guilt* for taking its suggestions to avoid work. Participant j4 described the feeling of taking Copilot's suggestions as "happily guilty," going so far as to thwart their chances of earning a performance bonus by deliberately wasting time.

"I was really wasting time... because I was feeling guilty for like putting all the questions in AI and just submitting like that, so I was like trying to take my time and see if I can understand it on my own" (j4)

Others also judged their own use of Copilot as excessive or immoral, noting, for example, how their professors might react if they were to witness their choices. Participants also engaged in *Self-Blaming* when they faced challenges using Copilot. As in the case of participant j16, they were quick to shift the blame away from Copilot itself.

"My understanding of [this problem] was kind of skewed... I think Copilot tried to the best of its ability, but... I gave the instructions and... that was more on me than on Copilot itself" (j16)

Copilot often failed to meet participants' needs, but they tended to take on the fault as their own. They felt that their ability to shepherd it toward a solution was lacking in some way, believing

that if only their prompts and manually written code were clearer or more comprehensive, Copilot would surely then be helpful.

6.5.3 Alienation

Alienation is the way Copilot estranges a programmer from their labor and discourages them from interacting with other people. The former includes alienation from the labor process, product, and self, while the latter is social *Alienation* [50].

Labor Alienation

Labor Alienation was when Copilot disconnected participants from the pleasure and identity they would otherwise find from programming alone, leading their programming labor to feel alien and unfamiliar. For example, participant i19 describes how their concept of programming as problem-solving was corrupted by a singular focus on guiding Copilot.

"I had to do less problem solving. It was more solving how to get the right answer out of Copilot rather than get the right answer out of myself." (i19)

More specific forms of *Labor Alienation* included feelings of low *Self-Efficacy*, lack of *Satisfaction*, and *Apathy*. When Copilot performed better than participants believed they could alone, this caused them to reflect poorly on their own abilities. For example, participant j20 was upset that Copilot could do the same work they could but faster.

"I could have probably [done this problem] but it did it faster than me, and that's what upset me... basically it didn't need me... I'm the dude that puts the stuff in the machine... and then the machine does all the work." (j20)

Constant reminders of Copilot's proficiency during the study wore on participants. For example, participant i2 ultimately regretted not making more effort to communicate to Copilot, concluding that their initial assumption they might know better was wrong.

If I had a do-over I would "try to leave more comments for GitHub Copilot to understand because I believe it has a better programming sense compared to me." (i2)

Once Copilot's effectiveness became clear, failures with it were taken to heart more seriously. Participant i42 implies that their failure to get unstuck while coding was even more frustrating and embarrassing because they had Copilot's assistance. After accepting their incompetence alone, accepting failure while having an advantage was not as easy.

I hate the feeling of being stuck and "I just couldn't continue, and I had like this robot helping me and I couldn't continue even with the robot" (i42)

Copilot also robbed participants of the *Satisfaction* of success. Participant i10 explained that they take comfort and enjoyment in the knowledge that they can solve a problem alone, which makes coding more fun. For i43, too, feeling accomplished came from overcoming a challenge without help from Copilot.

I'd rather try myself first before using Copilot because "I feel like it's more fun that way. I think it's also, like, I want to know that I can like figure it out on my own" (i10)

"I guess it's just a sense of accomplishment when you do it yourself." (i43)

Finally, three participants showed *Apathy*, a lack of care and investment, from using the AI. Participant j2 gave up during a solo task because they felt hopeless about being able to succeed without Copilot's help. This sentiment was echoed by j11 when they lost focus at an impasse, saying "parts of me didn't really care anymore." More concerning was j15's prediction about AI's impact on opinionation in programming.

"It will make people less dogmatic about like, the way that they choose to code... because they are outsourcing that style." (j15)

Their point is that when everyone begins to use Copilot and other AI code generators, code will begin to follow homogenous canonical conventions. They expect this to lead individuals to

abandon their personal convictions about which approaches are best, since whatever AI produces will be assumed to be best.

These different forms of *Labor Alienation* indicate that Copilot distances programmers from the enchantment and intellectualism that computing traditionally offered. Users feel pressured to use AI but then enter their abilities into a contest with it. It is embarrassing to fail with AI but not gratifying to succeed with it. As a result, some users lose interest in programming, because whatever meaning it had to them is missing.

Social Alienation

Social Alienation is when Copilot serves as a substitute for interacting with another person in the programming or educational process. Participants diverged about how AI stacked up to a human alternative. Participants j24 and i4 seemed perfectly fine with taking code and explanations from AI over a human.

Talking to Copilot "is the same as asking a teacher, like, 'hey, I'm stuck, how do I [do xyz]', but you don't have your teacher available" (j24)

I don't care whether AI or a human writes the code. "If it works the same, it's the same... if a human can explain it to me, I think I can also try to understand it by myself from the code that an AI produced" (i4)

On the other hand, not everyone agreed with settling for the convenience of the AI option. For participant i15, who used the older version of Copilot without the chat function, Copilot's assertive inline suggestions were not comparable to what they would expect from a human.

"I'd rather like talk to someone who can explain why they gave me that solution, rather than here's a solution and I have to, 'okay', pick it." (i15)

Other participants from the 'i' cohort, like i47, also indicated that asking questions about a solution was part of what made human interactions preferable. In the 'j' cohort, participants had access to Copilot Chat, so others, such as j2, felt similarly to j24 above; when it is inconvenient

to interact with another person, chatting with AI was a suitable substitute. Overall, it seems that AI code generators cause *Social Alienation* when they more closely approximate how humans act. Therefore, programmers can expect to become more separated from each other socially as AI and its interfaces continue to improve.

6.5.4 Rationalization

Rationalization is a value system that deems objective, calculable metrics as superior to intangible and subjective experiences. Productivity was the incentivized goal of the study, so using Copilot to be more productive was not relevant here (see *Pragmatism* instead). Rather, quotes reflecting *Rationalization* were those that showed a preexisting tendency for *Idolizing Technology*, seeing *Efficiency* as inherently good, or seeking out *Optimality* of code after its already being correct. These show ingrained capitalist ideals to glorify technology and numerically explainable perfection.

Idolizing Technology

Idolizing Technology was shown by many participants' overestimations of Copilot's abilities. They often expected Copilot to help them very effectively, reflecting a bias to assume technology is perfect prior to the study experience. Participant i1 found out during the study that Copilot was not as autonomous as they thought.

"I was kind of overestimating it." I was thinking, "if I were to just type a few instructions, basic instructions, it'd be able to do the rest for me, which it was not" (i1)

Another participant, j13, seemed to think Copilot had not just a good programming sense, but irrefutable opinions worthy of breaking a tie between misaligned collaborators. They suggested that Copilot would be useful at a hackathon so that a team could move forward with a decision based on its advice.

"With a hackathon you have 24 hours... and you're working with groups of other people, so let's say you and your teammates are having clashing ideas. You could consult

[Copilot] to just see which one it thinks works best” (j13)

The assumption that the opinions and actions of AI are superior to those of humans reflects that technology has an image of being deterministic and explainable. Ironically, AI is not, in fact, deterministic. It borrows the image of older forms of technology, causing a dangerous misperception. While devaluing the nondeterminism and uniqueness of humans, this misperception disguises the inhuman randomness of LLMs as a technological ideal.

Efficiency

Participants’ comments about Copilot’s speed indicated that they believed *Efficiency* to be inherently important. For example, participant i24 thought Copilot would be helpful in completing tedious tasks faster, implying that it is better to complete tasks quickly rather than slowly. This belief explains why other participants often applied an *Automation Bias* based on Pragmatism.

”It was a lot faster... with Copilot than it was without, so I think if you have something you’re trying to do, but you have a lot of more tedious things that you need to do in order to accomplish that, I think Copilot would be very helpful.” (i24)

Participant i43 similarly contrasted the popular fear of automation displacing programming jobs to their impression that displacing human workers is worthwhile if it improves the speed of production.

AI displacing jobs ”might be good in some ways if it makes things more efficient, but [it is] usually thought of as a bad thing.” (i43)

A mindset of *Rationalization* is so deeply rooted within computing that many readers will trivially concur that *Efficiency* is a worthwhile pursuit. However, other perspectives may consider manual attention to trivial activities and human fulfillment of labor needs to be preferable over efficiency. Somehow, the slowness and tedium of hand preparing a meal or custom-building furniture does not have an apparent parallel in the writing of computer software. If *Efficiency* is best, then provided AI is fast, *Automation Bias* takes over.

Optimality

Several participants also placed a value on *Optimality*, going beyond simply wanting code to work correctly toward desiring code which was perfect. For example, unlike those who wanted to produce code itself quickly, participant i2 thought Copilot could improve their learning by teaching them to write code that ran faster when executed. This reflects a desire to have code which is not only functionally correct but also fast at runtime. Participant i47 was ambivalent but also thought Copilot would be useful for writing code that ran more efficiently. Participant j20 most embodied the spirit of *Optimality*, asking Copilot to fix code that they already thought was correct.

"Sometimes I would highlight it and be like 'fix the code'... even if it didn't have a problem...I just wanted it to look a little bit better" (j20)

Along the same lines, participants j13 and i5 mentioned that Copilot could help to write more concise code, which they thought was better than longer code that worked the same way. Together, these examples point to a dimension of *Rationalization* that always leaves room for improvement. Users feel they need Copilot's help even when they do not, since they can always make their code better in some way.

6.6 Discussion

The goal of this study was to address the question: How do capitalist values influence novice programmers' relationship to code-generating AI within CS education? Evaluating this relationship requires situating the technology within the broader structures that shape it. Code-generating AI does not exist in a vacuum; it is embedded in pre-existing political-economic systems that shape its design, use, and the ways learners interact with it. Without such critical analysis, we risk overlooking the potential consequences of its deployment until those consequences have already materialized.

6.6.1 Rationalization & Automation Bias

We found that novices' values reflected capitalist trends of *Rationalization* and reverence for technology that triggered self-reinforcing *Automation Bias*. In capitalist economies, the logic of private accumulation encourages individuals to pursue self-interested actions that ultimately serve economic gain. As Weber argued, such preferences for outcomes that maximize material efficiency become standardized through bureaucratic systems. Once systematized to generate predictable revenue streams, these practices not only become the norm but are valued above alternatives that emphasize autonomy or creativity, because they so reliably deliver the economic returns embedded in the political economy. From a sociological perspective, this is the hallmark of rationalization: human action comes to be governed by calculability and efficiency rather than values of intrinsic meaning. In this context, technologies like Copilot are not tools but embodiments of rationalized systems.

Participants' immediate framing of Copilot as pragmatically useful reflects this preference and demonstrates the hold of instrumental rationality on novice programmers, who expressed a desire to simply complete the task. For many, it mattered little how the task was achieved so long as it was completed, underpinned by the assumption that task completion itself carries inherent value. Here, we see how the commodity form shapes educational labor: the exchange-value of finishing a task (grade, credential, or performance metric) overrides the use-value of programming as a learning process. The very structure of education under capitalism normalizes this substitution, positioning students to see efficiency itself as the end goal rather than knowledge, autonomy, or skill development. This phenomenon operates in two directions: while the political economy produces a preference for efficient outcomes, it also shapes the standards of computing labor that novices internalize as they prepare for future work. Participants frequently acknowledged that relying on Copilot might hinder their learning, since it performed the coding for them. Yet many coupled this recognition with the claim that such drawbacks mattered less for the "real world," where the labor market values speed and output over critical engagement or creative ingenuity. These dynamics reveal how students are positioned simultaneously as con-

sumers and producers within CS education. As consumers, they adopt AI tools like Copilot that are marketed on promises of efficiency, personalization, and convenience. Yet as producers, their labor—assignments, skills, and ultimately credentials—functions as capital that universities and employers commodify. Copilot collapses these roles: students consume automation while producing outputs that are valued less for their intrinsic meaning than for their exchange-value within educational and labor markets.

A prior qualitative study found that software engineers using neural machine translation subscribed to this idea, believing a working product mattered more than whether AI produced the code [86]. Our participants echoed this logic: they emphasized the importance of *Efficiency* and expressed a desire for code that approached *Optimality*—defined in terms of runtime speed, aesthetics, and conciseness. Notably, some framed this pursuit of *Optimality* through an uncritical trust in AI, treating it as an infallible source of expertise capable of delivering such outcomes. This tendency signaled a pre-existing inclination of *Idolizing Technology*. Several participants even spoke of Copilot in human-like terms, elevating its outputs as the ideal choices. Such responses illustrate how prevailing market values shape conceptions of AI: technology becomes revered under the sway of instrumental rationality, precisely because of its perceived ability to achieve economic ends with efficiency.

Placing Copilot onto a pedestal predisposed participants to apply an *Automation Bias* in their interactions with it. While novelty may have played a role, as suggested by prior AIDE studies [33], participants largely cited pragmatic reasons for relying on the tool. *Pragmatism* drove them to have Copilot take over and produce code for them, as it was very effective at doing so. Many believed Copilot to be effective, and, under conditions where they were incentivized to perform highly and allowed to use AI as much or as little as they wished, they often delegated tasks to it. This is not direct evidence of overreliance, but it does reveal how capitalist priorities—productivity and efficiency—shaped their responses. Participants frequently used Copilot even when they did not need to, indicating that when speed was paramount, automation became the preferred strategy. In a world where efficiency is equated with quality, it is unsurprising that participants came to idolize technology, especially the performative capabilities of Copilot.

This tendency cannot be dismissed as an artifact of experimental design. Prior studies have found similar patterns: participants requested full solutions from AI code generators [2, 39], accepted clearly flawed code [69], and used outputs in ways that revealed limited understanding [39]. Ritzer’s [65] theory of McDonaldization—an extension of Weber’s rationalization—helps explain this trend. All sectors of life, he argues, have come to prize efficiency, predictability, and control. Copilot embodies these logics. It delivers speedy, reliable outputs that reduce uncertainty, while also manifesting the fourth tenet of McDonaldization: control through non-human technology. In rationalized systems, people are viewed as the greatest sources of inefficiency [65]. From this perspective, companies adopt AI to reduce error and increase control, while students, as seen in our study, apply automation bias because they believe delegating to AI maximizes accuracy and minimizes mistakes in their own interest. Although this deference occurred in an experimental context, it likely reflects the attitudes novice programmers will carry into the labor market, where automation is prized for completing tasks quickly, reliably, and with minimal human error.

As with our participants, their bias manifested as *Passivity* within the interaction. They let Copilot take over control and, like in [83], sacrificed understanding and entered unrecoverable cycles of debugging. While this was sometimes initiated by *Voluntary* actions, Copilot’s triggerless suggestions also caused *Involuntary Passivity*. By seeing its suggestions before having a chance to think, users are biased by what they see and forced into a passive role. This is similar to the ideas of “incidental specification” discussed in [31] and Copilot’s intrusion on novices’ thought processes reported in [59] and [58]. Rather than users having the agency to make requests of AI, triggerless suggestions encourage prompts which are crafted as an unintentional byproduct of the coding process and interruptions which can be distracting and persuasive.

The influence of AI also leaves a lasting mark. When participants went back to programming alone, they suffered the constant *Temptation* to use Copilot instead. Knowing what it had to offer them, it was hard to shake the thought that any difficulties they encountered were unnecessary and avoidable. The tempting nature of Copilot’s suggestions for novices was reported in [59], but several other studies further explain the impact. One study of CS1 students found that par-

ticipants in a group with access to ChatGPT underutilized course materials, internet search, and peer forums compared to those in the control group [89]. Similarly, two prior studies found that CS students' use of AI code generators was bimodal, either using it heavily or very little at all [27, 48]. Together, these and our findings suggest that AIDEs create a black hole effect, where students are lured in by benefits that they then struggle to leave behind.

Finding 1 Novice programmers have preexisting values to favor programming goals that can be calculated numerically, tending also to favor technology to achieve these. By discovering GitHub Copilot's effectiveness for enhancing productivity, participants applied a bias for using AI that reduced autonomy and embedded a reinforcing temptation to do so repeatedly.

6.6.2 Individualism

We also found that novice programmers' beliefs about the responsibility of an AIDE user reflect individualistic capitalist values. In a capitalist economic system, an individual is expected to gain and lose capital according to their own choices. A neoliberal ideal would suggest that responsible decisions, those that produce valuable software with minimal negative externalities, are incentivized by a free market. Irresponsible decisions, such as those compromising safety by deprioritizing quality control, should then (theoretically) be punished by consumers' choosing other producers in the market or civil lawsuits that incur economic penalties. AIDEs like GitHub Copilot can produce code with security vulnerabilities [56], and this can lead programmers to write less secure code when using an AIDE [57]. Though AI is ostensibly to blame for such an impact on code quality, *Individualism* reflects a tendency to blame the human user instead. This echoes neoliberal logics of responsabilization, where systemic risks and responsibility shift from higher institutions to the individual, who must continually manage their own exposure to failure [67].

This was reflected by our participants' ascription of responsibility to themselves through the entire process of using Copilot. First, they felt they should apply *Self-Regulation* to decide

whether using Copilot was appropriate at all and then for deciding how much to use it. In neoliberal systems, self-regulation is not only a pragmatic skill but also a moral obligation: subjects are expected to internalize discipline and treat their own behavior as a site of investment. This illustrated what Marx [50] would describe as alienation—shifting of responsibility for structural conditions onto the laborer, even when the means of production introduce risks outside of the workers control. CS education researchers agree that students will need to self-monitor and self-regulate, but also emphasize the need for better metacognitive support to enhance the effectiveness of AIDEs for meeting learning objectives rather than productivity objectives [2, 58]. A study of CS students measured subjective self-regulation and self-reports of AI use, finding no relationship [48]. This indicates that though it is easy to expect students to regulate their use of AI, it may be that the design of AIDE interfaces is influencing them to use it ineffectively for learning [58].

Participants also felt they were responsible for *Shepherding*—understanding the task at hand and communicating it well to Copilot. Beyond the formalization of *Shepherding* as an interaction pattern by Prather et al. [59], several prior studies have noted the importance of an AIDE’s user taking on this role [20, 31, 39, 53]. What is less often expressed is a responsibility of the AIDE or its producer to effectively obtain the necessary information from the user. An example of this perspective is the Kiro¹ IDE, which uses AI in a specification-driven development paradigm that takes concrete steps to request the user’s intentions before generating content. The expectation for a user to guide and control the AIDE instead is an ironic one given AI’s tendency to degrade user autonomy. This expectation can be seen as part of rationalization [85], where tools designed to make work more efficient actually embed new procedural demands onto workers, narrowing their autonomy further. Individuals paradoxically bear heightened responsibility for outcomes even as actual decision-making authority shifts toward automated systems. However, in a neoliberal mindset, the AI creator has produced value by creating the AIDE, so the user is expected to take over responsibility by learning to use and adapt to it from there.

The final responsibility participants felt was one for taking *Ownership* of the code itself and

¹<https://kiro.dev/>

a knowledge about how it worked. This reflects Braverman's [11] analysis of deskilling: while technology simplifies some tasks, the worker is simultaneously burdened with accountability for the labor process they no longer fully control. A prior social media study noted that AI use by inexperienced programmers was deemed unacceptable in safety-critical domains, since the programmer must remain accountable for code reliability [20]. The extension of accountability for understanding of produced code was also reflected by the perspectives of CS students reported in [91]; several participants conveyed negative judgements about the idea of students using code from generative AI without being able to explain how or why that code was appropriate. Here again, capitalist individualism shows through, as code is considered a product that reflects implicit promises from the programmer. This baseline expectation motivates many software producers to deliver their code with licensing documents (e.g. the MIT² license) that explicitly disclaim responsibility for any issues a user may encounter with the software. In Marxist terms, this resembles the commodification of intellectual labor: the code is abstracted into a commodity that can be exchanged, while liability is pushed away from producers and onto individual users. Certainly, in CS education, the assumption of full understanding being necessary should be scrutinized [2]. In the past, understanding code fully was unavoidable for an effective programmer, but the capabilities of AIDEs potentially change this fact. If an explanation of code is truly needed for some maintenance or auditing purpose in the future, perhaps another AI-based tool could take on the task. In such a case, a student could be successful by producing useful code, even without understanding it. Considering code as a form of capital accrued by the programmer's labor perpetuates the idea that they should be held accountable for its every consequence. Instead, if code is merely a means for computers to communicate human values to one another, only the values originated by humans might be scrutinized.

Individualism also caused *Guilt and Self-Blaming* for our participants because of the broad Responsibilities they placed on themselves. Not only did participants feel they should self-regulate their AI usage and own an understanding of the code; they also felt that it was morally wrong not to do so. This aligns with concerns reported by students and educators in prior work about

²<https://opensource.org/license/mit>

AI's likelihood of contributing to academic integrity violations [5, 60, 91]. At the same time, CS educators recognize that the purpose of CS education may be changing as AI is adopted in industry [60]. The rational obsession of capitalism with quantifiable productivity metrics will drive adoption of AIDEs, reducing the need for programmers to know how to code manually. This means that our participants' feelings of guilt may be misplaced, causing unnecessary harm to them. This resembles a neoliberal subjective account, where individuals come to govern themselves through guilt when they cannot live up to market logics of productivity and accountability. This also mirrors the critique of education under neoliberalism [21], students are increasingly treated as entrepreneurial actors responsible for optimizing their own learning, even if structural conditions limit their agency.

Participants also blamed themselves when AI did not help them effectively, pointing to their failure to understand and guide Copilot as the root problem. Though many prior studies recognized the importance of the user's behavior, they did not all report such an effect on users' feelings. Users may not have blamed themselves directly, but they also avoided blaming the AI [27, 86]. However, when time pressure was imposed as we did here, participants did blame themselves [53]. Some participants in [59] also saw no downside to using Copilot when they used it without any time pressure. The takeaway is that performance-centric environments, which exist for educational assessments and time-bound projects, encourage AIDE users to blame themselves for productivity issues.

Finding 2 Novice programmers have individualistic mindsets that make them feel immense responsibility for using AI appropriately at every stage. They feel guilty for handing over control to AI, lose satisfaction when succeeding with it, and blame themselves when failing with it.

6.6.3 Alienation

We also found that Copilot alienated participants from the programming process and threatened their interest in future social activities. By fulfilling programming work, Copilot takes control away

from the user. Early work found that some programmers are averse to this process, desiring to maintain control of the interaction [83]. This is also unsurprising given the burdens users place on themselves (see Section 6.6.2); they cannot afford to give up control if they are liable for the consequences. As they let Copilot do more work, they lost touch with their own sense of value within the programming process, acting merely as users rather than creators. This resonates with Marx's [50] theory of alienation, in which workers become estranged from the product of their labor, from the process of production, from themselves, and from other workers. Copilot's intervention simultaneously detaches students from the code itself, from the act of coding, and from the collaborative relationships that programming education often relies on.

This was also reported in an early qualitative AIDE study, where programmers changed their commenting behaviors to adjust to Copilot [3]. Their participants were palpably enthusiastic about their changing role, but the tone from ours was one of disillusionment with Copilot and its growing abilities. They felt *Apathy* toward challenging problems and advocating for their opinions and described how producing correct code was less satisfying with Copilot than doing so alone. Here, we see what Braverman [11] calls "deskilling": the automation of complex labor reduces the worker to a monitor of the machine rather than an active problem solver, hollowing out opportunities for creativity and satisfaction. Whereas in human pair programming students' feelings of satisfaction increase [74], collaborating with AI is alienating. The comparison between AI and human partners also echoes the concept of rationalization: efficiency and predictability replace social bonds, with technical success prioritized over human meaning-making.

A related challenge is the blurring of lines between AI and other tools, which prior work has reported is a challenge for novices [59, 82]. If students may not even be sure whether they used AI to complete a task, their self-reflections may be harder to place. Anecdotally, we noticed some participants confusing the automated testing interface as being part of Copilot's functionality. Overall, though, they seemed to know when they were using Copilot.

Seeing how effective it was, participants compared themselves to Copilot unfavorably, harming their *Self-Efficacy*. This goes against prior work that suggests AIDEs can improve student *Self-Efficacy* [90], aligning better with evidence that AI use correlates with low *Self-Efficacy* [48]. It

may not be that AI simply increases or decreases *Self-Efficacy* directly but instead that it provides novel avenues for either change to occur [2]. Here we see that comparison is a clear avenue for *Self-Efficacy* harm. In CPE terms, this illustrates how capitalist competition structures identity: students measure their worth against the productivity of a machine, internalizing market logics that position them as failing entrepreneurs of their own labor-power. Students tie their sense of purpose and self-worth to productivity rather than craft or skill. If students fail to complete a task, they see themselves as bad coders; if they succeed on the task with Copilot, their sense of skill is undermined because the accomplishment belongs more to the tool than to them. This creates a lose-lose situation where failure feels personal and success diminishes their sense of identity as programmers. The broader implication is that the human role in coding is shrinking. Students start comparing themselves to AI—which they view as faster and more effective—and conclude that they cannot compete in a society obsessed with efficiency.

Finally, we noticed trends in participants' comments that place Copilot as a threat to social interactions with peers and instructors. We found mixed opinions about whether Copilot could replace a human interaction, but hesitancy seemed to be around perceived limitations of Copilot's features. Due to the rapid pace of advancement, participants at our two study sites used different versions of Copilot, with the second group having a chat feature available. It seemed that for participants who could get explanations from Copilot, the need for a human who could do the same was less necessary than for those participants who had only in-editor suggestions. Specifically, the ability for AIDEs to explain things in natural language has been an essential feature for user acceptance in prior studies of both students and professionals [2, 46, 69]. Limitations that have been reported by participants in the past are also steadily becoming commonplace among AIDEs like Copilot. For example, the introduction of agent behaviors to Copilot has enabled it to ingest outside resources for use in the programming context, something that users previously cited as a limitation [43]. Many studies have reported the tendency for users to anthropomorphize AIDEs [20, 59, 69, 86], and this stands only to continue as feature sets expand into tasks previously considered as inherently irreplaceable. Whereas [31] suggested participants may have distrusted AI generated code as compared to that written by instructors, our participants did not

seem to think so. Peer and instructor relationships lose value unless they can provide a comparative advantage over automation. Therefore, the affective and social dimensions of education risk being hollowed out, valued only insofar as they enhance efficiency. We argue that their sentiment originates from the structure of a capitalist economy. Labor is the origination of capital itself, so accumulating capital depends on human labor. Rather than considering the interaction among humans as valuable within itself, the capitalist framing says that human interactions are only valuable to the extent they outperform the alternative to use automation.

Finding 3 AI exacerbates capitalism’s alienation of programmers from labor and human relationships because it diminishes the traditional humanity of writing and understanding code.

6.7 Educational Implications & Recommendations

To counteract the harms of code generating AI, CS education should respond by encouraging students to challenge their own assumptions about what goals they have for programming endeavors. Overall, we consider the goal of computer science to be using computers to promote human welfare. Our findings show that capitalist values pervade CS students’ perspectives about programming and using AI code generators. We observed patterns of *Rationalization* and *Automation Bias* by which participants prioritized productivity and perfection, preferring an AIDE for achieving these over programming alone. Resulting harms were that participants felt guilty for using GitHub Copilot, inadequate in comparison to it, and responsible for negative outcomes but not positive ones. They felt alienated from programming and revealed ways Copilot will threaten social relationships by assuming a pseudo-social role. If CS students apply a mindset to use AI in pursuit of calculable goals, they may promote human welfare in parallel to capital accumulation. However, by engaging with opposing perspectives, students can form more nuanced and holistic approaches to using AI in their computing education and careers. Next, we propose intentionally contrarian perspectives to our findings. We define these not to claim they are correct or superior

to capitalist ideals, but merely to give educators a concrete Socratic basis for challenging vague and latent assumptions. Each of these perspectives can be applied in nontechnical instruction, an important but often overlooked aspect of programming education.

6.7.1 Challenge Rationalization

Participants were unsurprisingly influenced by a dominant value to glorify coding quickly to produce fast-running, optimal solutions to problems. While at first, this seems apparently in line with a goal to promote human welfare, the perspective is myopic to the relative importance of perfection in a given context. There are many cases where metrics for productivity fail to correlate to true value to humanity. Marx emphasizes that capitalism promotes the exchange-value of a product rather than its true use-value [49]. As an example, developers in a corporate setting might be encouraged to work through project management tickets quickly as a measure of productivity. If a developer works on a valuable task that is time-consuming, but only completes a single ticket, the ticket speed metric could give an inappropriately poor representation of their contributions, even in an economic sense. Whereas the exchange-value of the work to clients might be low in that ticket resolution time would be high, the use-value to the client might still be high given that the developer made an important contribution.

This concept extends to investments by students in time-consuming educational programming. As our participants discovered, programming with Copilot gave a clear performance advantage while working under time pressure. However, the presumably lower engagement with the work they complete with AI can lead to poor representation of the endeavor's success. Perhaps some concept learned in a slow solo programming experience as a student can translate several years later to the programmer developing a life-saving advancement to the firmware in a medical device. In the short term, it seems that the student has covered the same material in a longer time period, making working slower worse. In reality, the investment of time as a student is clearly valuable when considered at a systems level.

6.7.2 Promote Communal Perspectives

Participants felt they had responsibility as users of code-generating AI for how much to use it, how to use it, the code resulting, and the consequences of that code. Notably, neither the vendors of user-facing AI products nor the maintainers of backend AI infrastructure were discussed in this chain of responsibility. Whereas capitalist *Individualism* leaves a CS student liable for their behavior in relation to AIDEs, a communal perspective cautions other actors in the society to avoid behaviors that set users up for failure. For example, GitHub Copilot's student discount gives CS students an opportunity to use a premium AIDE for free. This has the benefit to help students learn about AIDEs and gain any learning benefits but can also harmfully broaden the reach of the *Alienation* we identified. Furthermore, the design of AIDEs has productivity in mind. Though certain features, such as system prompts or "instructions" as they are named within Copilot's interface, may be useful for CS students to better align their desire to learn to an AIDE's behavior, vendors do not take responsibility for encouraging users to take advantage of such features. Discussion with students on these topics will be important as they transition from AI consumers to AI producers. Education that considers broader purposes for using AIDEs can then incite later generations of the technology to represent additional values beyond only capitalist ones.

6.7.3 Emphasize Developer Experience

Copilot's positioning to alienate CS students from themselves, the labor of programming, and their social connections represents a deprioritization of their developer experience (DX). Though human-centered computing often promotes human welfare by upholding user experience (UX), it sometimes fails to consider the importance of DX. Just as burnout and dissatisfaction of medical professionals can be harmful to the patients their labor supports, dissatisfied programmers are impaired with respect to creating value for humanity. Additionally, finding purpose and self-efficacy within programming is a vital prerequisite for students to persist through education to the point of being strong contributors later on. As such, it is vital that educators stress to students that they deserve to find enjoyment, satisfaction, confidence, and social connections within their

work, even if it costs time. Accessing useful AIDEs is becoming easier by the year, such that the time and effort associated with solving a problem alone or traveling to an instructor's office space to take a face-to-face meeting seem impractical. Students must be taught to hold simultaneously that using AI can enhance their programming but prioritizing themselves as humans is an upstanding tradeoff. This means they should take license to work together with peers over AI, accept worse code that they are proud to have produced without AI, and treasure inefficiently gained knowledge for its unrealized future benefits.

6.8 Threats to Validity

Our study design may be considered a contrived scenario in that participants were encouraged to be as productive as possible with an AI assistant known to be performant on the tasks they completed. This would tend to amplify apparent automation bias and worsen alienation and negative individualistic thought patterns. Still, our results expressed capitalist ideals beyond what the study design alone could explain. Regarding generalizability of the study tasks, we note that programmers may find more active participation, meaning, and self-efficacy within complex real-world problems, for which AI is less helpful. This would reduce reliance on AI and associated feelings of guilt, too. However, as AI improves, the exaggerated experiences we captured will likely transfer to increasingly difficult problem spaces as the practical need for human involvement decreases. This study is not highly generalizable to non-U.S. economies, although globalization promotes U.S. economic values across the world and several of our participants had non-U.S. national origins. The results are most applicable to the post-secondary educational culture at public universities in the eastern U.S. As for the validity of our method, we acknowledge our subjective lens but maintain that our systematic multi-rater process delivered an honest representation of our participants' experiences. We made efforts to represent as many of the participants' voices as possible and capture the breath of their experiences. While our findings provide little direct evidence regarding how capitalist values and AIDEs interact to harm CS students' learning, we encourage future work to explore how the hazards we identified may be counterproductive to the

mission of CS education.

6.9 Conclusion

This study used Marxist and Weberian theories to identify the harms of AI code generators to students within the market-driven CS education pipeline of the United States. We identified three key findings: (1) measurable programming goals were prioritized, with a preference for using AI to achieve them, (2) programmers were considered personally responsible for every aspect of programming with AI, (3) code-generating AI alienates programmers from their labor and their social connections. We recommend that CS education trains students to be AI-literate rather than AI-dependent by exploring pedagogies that critically and strategically integrate AI. This involves not promoting, but merely presenting, counter-capitalist ideology into nontechnical dialogs within CS classrooms.

References

- [1] Andrew Abbott. 2007. *Chaos of disciplines*. (Nachdr. ed.). Univ. of Chicago Press, Chicago, Ill. 259 pp. isbn: 978-0-226-00101-2 978-0-226-00100-5.
- [2] Matin Amoozadeh, Daye Nam, Daniel Prol, Ali Alfageeh, James Prather, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Student-AI interaction: a case study of CS1 students. (Oct. 11, 2024). arXiv: 2407.00305[cs]. doi:10.48550/arXiv.2407.00305.
- [3] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: how programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7, 85–111, OOPSLA1. Number: OOPSLA1.
- [4] Kathy Baxter, Catherine Courage, and Kelly Caine. 2015. *Understanding your users: a practical guide to user research methods*. Morgan Kaufmann. Retrieved Aug. 28, 2024 from <https://books.google.com/books?hl=en&lr=&id=l9-cBAAQBAJ&oi=fnd&pg=PP1&dq=Understanding+your+users:+A+practical+guide+to+user+research+methods&ots=dEIEZAXoRZ&sig=4ckI45L5qds4FShqQmSV-u-KsFO>.

- [5] 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*, 500–506.
- [6] Ted Benton. 2011. *Philosophy of social science: the philosophical foundations of social thought*. Ian Craib, (Ed.) (2nd ed., 10th anniversary ed ed.). *Traditions in social theory*. Palgrave Macmillan, Houndsmill, Basingstoke, Hampshire ; New York. 260 pp. isbn: 978-0-230-24260-9 978-0-230-24259-3.
- [7] Margaret M. Blair and Steven M. H. Wallman. 2001. *Unseen wealth: report of the Brookings Task Force on Intangibles*. In collab. with Brookings Task Force on Intangibles. Brookings Institution Press, Washington, D.C. 1 p. isbn: 978-0-8157-0113-2 978-0-8157-9125-6.
- [8] Pierre Bourdieu. 1977. Outline of a theory of practice. In *The new social theory reader*. Routledge, 80–86. Retrieved Aug. 12, 2025 from <https://www.taylorfrancis.com/chapters/edit/10.4324/9781003060963-11/outline-theory-practice-pierre-bourdieu>.
- [9] Pierre Bourdieu. 1998. *Practical reason: On the theory of action*. Stanford University Press. Retrieved Aug. 12, 2025 from <https://books.google.com/books?hl=en&lr=&id=2xFbVv-EHf4C&oi=fnd&pg=PR7&dq=Pierre+Bourdieu.+1998.+Practical+reason:+On+the+theory+of+action.+Stanford+University+Press.&ots=EYeukrvujD&sig=qIZSTz1ydFBaKh5gXpYkF0crxa4>.
- [10] Pierre Bourdieu. 1986. The forms of capital. In *The sociology of economic life*. (Third edition ed.). Mark S. Granovetter and Richard Swedberg, (Eds.) Routledge, Taylor & Francis Group, New York London. isbn: 978-0-429-49433-8.
- [11] Harry Braverman. 1998. *Labor and monopoly capital: the degradation of work in the twentieth century*. (25th anniversary ed. ed.). OCLC: 823170284. Monthly Review Press, New York. 338 pp. isbn: 978-1-58367-375-1.
- [12] Wendy Brown. 2017. *Undoing the demos: neoliberalism's stealth revolution*. (First paperback edition, this edition corrects errors in previous printings ed.). *Near futures*. Zone Books, New York. 295 pp. isbn: 978-1-935408-54-3.
- [13] Nancy Cartwright and Eleonora Montuschi. 2014. *Philosophy of social science: a new introduction*. Oxford university press, Oxford. isbn: 978-0-19-964509-1 978-0-19-964510-7.
- [14] Ron Eglash, Audrey Bennett, Laquana Cooke, William Babbitt, and Michael Lachney. 2021. Counter-hegemonic computing: toward computer science education for value generation and emancipation. *ACM Trans. Comput. Educ.*, 21, 4, (Oct. 15, 2021), 29:1–29:30. doi:10.1145/3449024.

- [15] John Fantuzzo. 2015. A course between bureaucracy and charisma: a pedagogical reading of max weber's social theory. *Journal of Philosophy of Education*, 49, 1, (Feb. 2015), 45–64. doi:10.1111/1467-9752.12095.
- [16] 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 (ACE '22)*. Association for Computing Machinery, New York, NY, USA, (Feb. 14, 2022), 10–19. isbn: 978-1-4503-9643-1. doi:10.1145/3511861.3511863.
- [17] 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 (ACE '23)*. Association for Computing Machinery, New York, NY, USA, (Jan. 30, 2023), 97–104. isbn: 978-1-4503-9941-8. doi:10.1145/3576123.3576134.
- [18] Nicholas Gardella, Raymond Pettit, and Sara L. Riggs. 2024. Performance, workload, emotion, and self-efficacy of novice programmers using AI code generation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2024: Innovation and Technology in Computer Science Education*. ACM, Milan Italy, (July 3, 2024), 290–296. isbn: 979-8-4007-0600-4. doi:10.1145/3649217.3653615.
- [19] Paul Du Gay, (Ed.) *The Values Of Bureaucracy*. Oxford University PressOxford, (Mar. 17, 2005). isbn: 978-0-19-927545-8 978-1-383-04182-8. doi:10.1093/oso/9780199275458.001.0001.
- [20] Gian Luca Scoccia. 2023. Exploring early adopters' perceptions of ChatGPT as a code generation tool, (Sept. 11, 2023). MAG ID: 4388214696. doi:10.1109/asew60602.2023.00016.
- [21] Henry A. Giroux. 2019. *Neoliberalism's War on Higher Education*. (2nd ed ed.). Haymarket Books, La Vergne. 1 p. isbn: 978-1-60846-334-3 978-1-64259-092-0.
- [22] GitHub. 2023. GitHub copilot · your AI pair programmer. GitHub. Retrieved July 24, 2023 from <https://github.com/features/copilot>.
- [23] Jonathan Hardy. 2014. *Critical Political Economy of the Media: An Introduction*. (1st ed.). Routledge, (June 20, 2014). isbn: 978-0-203-13622-5. doi:10.4324/9780203136225.
- [24] David Hartley. 1995. The 'McDonaldization' of higher education: food for thought? *Oxford Review of Education*, 21, 4, 409–423. Retrieved July 9, 2025 from <https://www.jstor.org/stable/1050719>.
- [25] Jonathan E. Haskel and Stian Westlake. 2018. *Capitalism without capital: the rise of the intangible economy*. Princeton university press, Princeton. isbn: 978-1-4008-8832-0 978-0-691-17503-4.

- [26] Toni Honicke and Jaclyn Broadbent. 2016. The influence of academic self-efficacy on academic performance: a systematic review. *Educational research review*, 17, 63–84. Retrieved Dec. 29, 2023 from <https://www.sciencedirect.com/science/article/pii/S1747938X15000639>.
- [27] Irene Hou, Sophia Mettill, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The effects of generative AI on computing students' help-seeking preferences. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*. Association for Computing Machinery, New York, NY, USA, (Jan. 29, 2024), 39–48. isbn: 979-8-4007-1619-5. doi:10.1145/3636243.3636248.
- [28] Walter Humes. 2022. THE 'IRON CAGE' OF EDUCATIONAL BUREAUCRACY. *British Journal of Educational Studies*, 70, 2, (Mar. 4, 2022), 235–253. doi:10.1080/00071005.2021.1899129.
- [29] Ursula Huws. 2014. *Labor in the Global Digital Economy: The Cybertariat Comes of Age*. Google-Books-ID: C98VCgAAQBAJ. NYU Press, (Dec. 5, 2014). 208 pp. isbn: 978-1-58367-463-5.
- [30] Yujeong Hwang, Anjali Das, Jane Waite, and Sue Sentance. 2023. Using a sociological lens to investigate computing teachers' culturally responsive classroom practices. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23)*. Vol. 1. Association for Computing Machinery, New York, NY, USA, (Sept. 10, 2023), 206–221. isbn: 978-1-4503-9976-0. doi:10.1145/3568813.3600112.
- [31] 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*, 1–15.
- [32] Richard Jenkins. 2000. Disenchantment, enchantment and re-enchantment: max weber at the millennium. *Max Weber Studies* 1, 1, 11–32.
- [33] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J. Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–19.
- [34] Martin Jonsson and Jakob Tholander. 2022. Cracking the code: co-coding with AI in creative programming education. In *Proceedings of the 14th Conference on Creativity and Cognition (C&C '22)*. Association for Computing Machinery, New York, NY, USA, (June 20, 2022), 5–14. isbn: 978-1-4503-9327-0. doi:10.1145/3527927.3532801.
- [35] Gregor Jošt, Viktor Taneski, and Sašo Karakatič. 2024. The impact of large language models on programming education and student learning outcomes. *Applied Sciences*, 14, 10, (Jan. 2024), 4115. Number: 10. doi:10.3390/app14104115.

- [36] Maria Kallia and Quintin Cutts. 2021. Re-examining inequalities in computer science participation from a bourdieusian sociological perspective. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021)*. Association for Computing Machinery, New York, NY, USA, (Aug. 17, 2021), 379–392. isbn: 978-1-4503-8326-4. doi:10.1145/3446871.3469763.
- [37] Evgeny Kamensky. 2017. Society. personality. technologies: social paradoxes of industry 4.0. *Економічний часопис - XXI*, 164, 3, 9–13. Retrieved Sept. 26, 2025 from <https://www.cceol.com/search/article-detail?id=546662>.
- [38] 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, (Apr. 19, 2023), 1–23. isbn: 978-1-4503-9421-5. doi:10.1145/3544548.3580919.
- [39] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–12. isbn: 979-8-4007-1653-9. doi:10.1145/3631802.3631806.
- [40] I.V. Kazmina, O.I. Muratova, T.V. Schegoleva, N.L. Volodina, and S.A. Slinchuk. 2020. The influence of modern information technologies on the efficiency of high-tech companies' performance. In *Proceedings of the Russian Conference on Digital Economy and Knowledge Management (RuDECK 2020)*. Russian Conference on Digital Economy and Knowledge Management (RuDECK 2020). Atlantis Press, Voronezh, Russia. doi:10.2991/aebmr.k.200730.051.
- [41] Amy J. Ko, Francisco Castro, Aakash Gautam, Anne Drew Hu, Sara Kingsley, Michael Lachney, and Aman Yadav. 2024. Computing, education, and capitalism. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, (Mar. 15, 2024), 1914. isbn: 979-8-4007-0424-6. doi:10.1145/3626253.3635362.
- [42] Michael Lachney, Jean Ryoo, and Rafi Santo. 2021. Introduction to the special section on justice-centered computing education, part 1. *ACM Trans. Comput. Educ.*, 21, 4, (Dec. 6, 2021), 25:1–25:15. doi:10.1145/3477981.
- [43] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A large-scale survey on the usability of AI programming assistants: successes and challenges. In *Proceedings of the IEEE/ACM 46th International*

- Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3608128.
- [44] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. Koli Calling '23: 23rd Koli Calling International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 13, 2023), 1–11. isbn: 979-8-4007-1653-9. doi:10.1145/3631802.3631830.
- [45] Yvonna S. Lincoln and Egon G. Guba. 1985. *Naturalistic inquiry*. Vol. 75. Sage. Retrieved Aug. 28, 2024 from <https://books.google.com/books?hl=en&lr=&id=2oA9aWlNeooC&oi=fnd&pg=PA7&dq=lincoln+and+guba+naturalistic&ots=0v1xYcVfvm&sig=0g8kKlMXyrx54UVh08HGQdrXLZI>.
- [46] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024. Teaching CS50 with AI: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 750–756. isbn: 979-8-4007-0423-9. doi:10.1145/3626252.3630938.
- [47] Lauri Malmi et al. 2014. Theoretical underpinnings of computing education research: what is the evidence? In *Proceedings of the tenth annual conference on International computing education research (ICER '14)*. Association for Computing Machinery, New York, NY, USA, (July 28, 2014), 27–34. isbn: 978-1-4503-2755-8. doi:10.1145/2632320.2632358.
- [48] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-regulation, self-efficacy, and fear of failure interactions with how novices use LLMs to solve programming problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2024: Innovation and Technology in Computer Science Education*. ACM, Milan Italy, (July 3, 2024), 276–282. isbn: 979-8-4007-0600-4. doi:10.1145/3649217.3653621.
- [49] Karl Marx. 1867. *Capital: Critique of Political Economy, Volume 1*. Paul Reitter and Paul North, (Eds.) Princeton University Press. isbn: 978-0-691-24046-6. doi:10.1515/9780691240466.
- [50] Karl Marx. 2023. Economic and philosophic manuscripts of 1844. In *Social theory re-wired*. Routledge, 142–147. Retrieved Mar. 11, 2025 from <https://www.taylorfrancis.com/chapters/edit/10.4324/9781003320609-19/economic-philosophic-manuscripts-1844-karl-marx>.

- [51] Kathleen L. Mosier and Linda J. Skitka. 1999. Automation use and automation bias. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 43, 3, (Sept. 1999), 344–348. doi:10.1177/154193129904300346.
- [52] Marcello Musto, (Ed.) Marx for today. OCLC: 988792943. Routledge, London ; New York. 239 pp. isbn: 978-1-135-70040-9.
- [53] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24: IEEE/ACM 46th International Conference on Software Engineering. ACM, Lisbon Portugal, (Apr. 12, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3639187.
- [54] Emil Øversveen. 2022. Capitalism and alienation: towards a marxist theory of alienation for the 21st century. *European Journal of Social Theory*, 25, 3, (Aug. 2022), 440–457. doi:10.1177/13684310211021579.
- [55] Raja Parasuraman and Dietrich H. Manzey. 2010. Complacency and bias in human use of automation: an attentional integration. *Human Factors*, 52, 3, (June 1, 2010), 381–410. doi:10.1177/0018720810376055.
- [56] 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.
- [57] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, (Nov. 21, 2023), 2785–2799. isbn: 979-8-4007-0050-7. doi:10.1145/3576915.3623157.
- [58] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: the benefits and harms of generative AI for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*. ICER 2024: ACM Conference on International Computing Education Research. ACM, Melbourne VIC Australia, (Aug. 12, 2024), 469–486. isbn: 979-8-4007-0475-8. doi:10.1145/3632620.3671116.
- [59] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. “it’s weird that it knows what i want”: usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 29, 2024), 1–31. doi:10.1145/3617367.

- [60] James Prather et al. 2023. The robots are here: navigating the generative AI revolution in computing education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education* (ITiCSE-WGR '23). Association for Computing Machinery, New York, NY, USA, (Dec. 28, 2023), 108–159. isbn: 979-8-4007-0405-5. doi:10.1145/3623762.3633499.
- [61] Yolanda A. Rankin, Jakita O. Thomas, and Sheena Erete. 2021. Black women speak: examining power, privilege, and identity in CS education. *ACM Transactions on Computing Education*, 21, 4, (Oct. 25, 2021), 26:1–26:31. doi:10.1145/3451344.
- [62] David Ricardo. 1817. *On the Principles of Political Economy and Taxation*. Cambridge library collection. Cambridge University Press, Cambridge. isbn: 978-1-107-58942-1. Retrieved Aug. 12, 2025 from <https://proxy1.library.virginia.edu/login?url=https://doi.org/10.1017/CB09781107589421>.
- [63] Kerstin Rieder. 2010. (PDF) the working customer – an emerging new type of consumer. ResearchGate. (Jan. 2010). Retrieved Sept. 26, 2025 from https://www.researchgate.net/publication/257266524_The_Working_Customer_-_an_Emerging_New_Type_of_Consumer.
- [64] Daniel Rigney. 2010. *The Matthew Effect: How Advantage Begets Further Advantage*. Google-Books-ID: TCiCwj1UFI4C. Columbia University Press, (Feb. 23, 2010). 176 pp. isbn: 978-0-231-52040-9.
- [65] George Ritzer. 2021. *The McDonaldization of society: into the digital age*. (Tenth edition ed.). SAGE, Los Angeles. 217 pp. isbn: 978-1-5443-9801-3.
- [66] George Ritzer and Steven Miles. 2019. The changing nature of consumption and the intensification of McDonaldization in the digital age. *Journal of Consumer Culture*, 19, 1, (Feb. 2019), 3–20. doi:10.1177/1469540518818628.
- [67] Nikolas S. Rose. 2012. *Powers of freedom: reframing political thought*. (9. reprint ed.). Cambridge Univ. Press, Cambridge. 321 pp. isbn: 978-0-521-65905-5.
- [68] Alexander Rosenberg. 2018. *Philosophy of Social Science*. (5th ed.). Routledge, New York, (Apr. 17, 2018). 360 pp. isbn: 978-0-429-49484-0. doi:10.4324/9780429494840.
- [69] 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. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. IUI '23: 28th International Conference on Intelligent User Interfaces. ACM, Sydney NSW Australia, (Mar. 27, 2023), 491–514. isbn: 979-8-4007-0106-1. doi:10.1145/3581641.3584037.
- [70] Emma Rowe. 2019. Capitalism without capital: the intangible economy of education reform. *Discourse: Studies in the Cultural Politics of Education*, 40, 2, (Mar. 4, 2019), 271–279. doi:10.1080/01596306.2019.1569883.

- [71] Isaak Ilyich Rubin. 1973. *Essays on Marx's Theory of Value*. OCLC: 1457057588. Pattern Books. isbn: 978-3-0340-0472-5.
- [72] Caitlin Sadowski and Thomas Zimmermann. 2019. *Rethinking productivity in software engineering*. Springer Nature. Retrieved Sept. 9, 2024 from <https://library.oapen.org/handle/20.500.12657/22839>.
- [73] Jean Salac, Lena Armstrong, F. Megumi Kivuva, Jayne Everson, and Amy J. Ko. 2024. How economically marginalized adolescents of color negotiate critical pedagogy in a computing classroom. *ACM Trans. Comput. Educ.*, 25, 1, (Dec. 17, 2024), 2:1–2:29. doi:10.1145/3702330.
- [74] Norsaremah Salleh, Emilia Mendes, and John Grundy. 2011. Empirical studies of pair programming for CS/SE teaching in higher education: a systematic literature review. *IEEE Transactions on Software Engineering*, 37, 4, (July 2011), 509–525. doi:10.1109/TSE.2010.59.
- [75] Abdulhadi Shoufan. 2023. Can students without prior knowledge use ChatGPT to answer test questions? an empirical study. *ACM Trans. Comput. Educ.*, 23, 4, (Dec. 11, 2023), 45:1–45:29. doi:10.1145/3628162.
- [76] Adam Smith. 1776. *An Inquiry Into the Nature and Causes of the Wealth of Nations*. Printed for W. Strahan and T. Cadell, London. Retrieved Mar. 19, 2025 from https://proxy1.library.virginia.edu/login?url=http://galenet.galegroup.com/servlet/Sabin?af=RN&ae=CY104116401&srchtp=a&ste=14&q=viva_uva.
- [77] Nick Srnicek. 2017. *Platform capitalism. Theory redux*. Polity, Cambridge, UK Malden, MA. 1 p. isbn: 978-1-5095-0488-6 978-1-5095-0490-9 978-1-5095-0489-3.
- [78] Andreas Kjær Stage and Ea Høg Utoft. 2022. Starting grant with double-blinded review: an escape from the production line of big science? (Mar. 9, 2022). doi:10.31235/osf.io/cvkmw.
- [79] Nicola Taylor. 2004. Reconstructing marx on money and the measurement of value. In *The Constitution of Capital: Essays on Volume 1 of Marx's Capital*. Palgrave Macmillan UK, London. isbn: 978-1-4039-3864-0.
- [80] 2014. *The culturally engaging campus environments (CECE) model: a new theory of success among racially diverse college student populations*. Samuel D. Museus. *Higher Education: Handbook of Theory and Research*. Springer Netherlands, Dordrecht, 189–227. isbn: 978-94-017-8004-9 978-94-017-8005-6. doi:10.1007/978-94-017-8005-6_5.
- [81] U.S. Dept. of Commerce, National Telecommunications and Information Administration. 1999. Falling through the net: defining the digital divide. Retrieved Feb. 27, 2025 from http://security.cc.columbia.edu/itc/polisci/W3923/digital_divide.pdf.

- [82] Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: integrating LLMs into CS1 instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (ITiCSE 2024). Association for Computing Machinery, New York, NY, USA, (July 3, 2024), 297–303. isbn: 979-8-4007-0600-4. doi:10.1145/3649217.3653584.
- [83] 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* (CHI EA '22). Association for Computing Machinery, New York, NY, USA, (Apr. 28, 2022), 1–7. isbn: 978-1-4503-9156-6. doi:10.1145/3491101.3519665.
- [84] Tina Vrieler and Minna Salminen-Karlsson. 2022. A sociocultural perspective on computer science capital and its pedagogical implications in computer science education. *ACM Trans. Comput. Educ.*, 22, 4, (Sept. 15, 2022), 44:1–44:23. doi:10.1145/3487052.
- [85] Max Weber, Peter R. Baehr, Gordon C. Wells, and Max Weber. 2012. *The Protestant ethic and the "spirit" of capitalism and other writings*. (23. print ed.). *Penguin twentieth-century classics*. Penguin Books, New York. 392 pp. isbn: 978-0-14-043921-2.
- [86] Justin D. Weisz et al. 2021. Perfection not required? human-AI partnerships in code translation. In MAG ID: 3154248444. (Apr. 14, 2021), 402–412. doi:10.1145/3397481.3450656.
- [87] Brian Whitworth. 2005. Polite computing. *Behaviour & Information Technology*, 24, 5, (Sept. 2005), 353–363. doi:10.1080/01449290512331333700.
- [88] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE code generation from natural language: promise and challenges. *ACM Transactions on Software Engineering and Methodology*, 31, 2, (Mar. 4, 2022), 29:1–29:47. doi:10.1145/3487569.
- [89] Yuankai Xue, Hanlin Chen, Gina R. Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT help with introductory programming?an experiment of students using ChatGPT in CS1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (ICSE-SEET '24). Association for Computing Machinery, New York, NY, USA, (May 24, 2024), 331–341. isbn: 979-8-4007-0498-7. doi:10.1145/3639474.3640076.
- [90] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, 4, (Jan. 1, 2023), 100147. doi:10.1016/j.caeai.2023.100147.

- [91] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in computing education: perspectives of students and instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*. 2023 IEEE Frontiers in Education Conference (FIE). (Oct. 2023), 1–9. doi:10.1109/FIE58773.2023.10343467.

Chapter 7

Fast and Forgettable: A Controlled Study of Novices' Performance, Learning, Workload, and Emotion in AI-Assisted and Human Pair Programming Paradigms

Chapter Preamble

This chapter carries out Aim 3, the final empirical investigation in this dissertation, through an experiment comparing the assistance of GitHub Copilot to a social human-human pair programming alternative. The findings of Chapters 2, 3, 5 and 6 suggested that emotional connection might be lacking in novice-AIDE interactions. Pair programming between humans has been a useful approach in both professional and educational settings for many years. AIDEs are often conceptualized as partners or assistants, but it is unclear whether the benefits of human programming

Authorship, Reuse, and Licensing Statement This chapter was coauthored with James Prather , Paul Denny , Juho Leinonen , Raymond Pettit , and Sara L. Riggs , who consented to the inclusion of this chapter in this dissertation. Its contents may appear in a future ACM publication (see Preprint for Chapter 7 in Appendix E). The license for that paper applies to this chapter. The fallback license is listed on the copyright page for this dissertation. ACM allows complete reuse of published papers in doctoral dissertations per the following section of the *ACM Author Rights* policy, retrieved on March 16, 2026: "Authors can include partial or complete papers of their own in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included."

partners are similarly provided by AIDEs. The work described here explores this issue by comparing human and AI teammates, focusing on initial team performance, workload and emotion, and individual retest performance (a measure of learning). The findings are then explained in relation to the Control-Value Theory of emotion.

Abstract

Code-generating Artificial Intelligence has gained popularity within both professional and educational programming settings over the past several years. While research and pedagogy are beginning to cope with this change, computing students are left to bear the unforeseen consequences of AI amidst a dearth of empirical evidence about its effects. Though pair programming between students is well studied and known to be beneficial to self-efficacy and academic achievement, it remains underutilized and further threatened by the proposition that AI can replace a human programming partner. In this paper, we present a controlled pair programming study with 22 participants who wrote Python code under time pressure in teams of two and individually with GitHub Copilot for 20 minutes each. They were incentivized by bonus compensation to balance performance with understanding and were retested individually on the programming tasks after a retention interval of one week. Subjective measures of workload and emotion as well as objective measures of performance and learning (retest performance) were collected. Results showed that participants performed significantly better with GitHub Copilot than their human teammate, and several dimensions of their workload were significantly reduced. However, the emotional effect of the human teammate was significantly more positive and arousing as compared to working with Copilot. Furthermore, there was a nonsignificant absolute retest performance reduction in the AI condition and a larger retest performance decrement in the AI condition. We recommend that educators strongly consider revisiting pair programming as an educational tool in addition to embracing modern AI.

7.1 Introduction

Generative Artificial Intelligence (GenAI) powers a number of tools programmers and computing students can use to assist in writing code, solving problems, and completing other coding related tasks [23]. These include general-purpose chatbots like ChatGPT and editor-based coding toolsets like GitHub Copilot [12], the focus of this work. Copilot is branded as an "AI pair programmer," a nod to a team programming methodology popularized in the early 2000s called "pair programming" [7, 28]. Because AI-powered behaviors can resemble human behaviors, GenAI is often framed as a human-like assistant that can help with work otherwise done alone or with another human. Though research has begun to compare the new AI-assisted paradigm to traditional solo and pair programming paradigms, the vast breadth of potential harms and benefits still remains largely unexplored [62].

AI assistance poses special threats to computing students. A burgeoning ecosystem of educational AI might potentially help with learning, but commonplace tools developed for professionals (i.e., GitHub Copilot) may not [62, 61]. In general, code-generating AI improves basic programming performance, but programmers can become reliant on its help [40, 8]. Then, when AI becomes unavailable or makes mistakes, programmers may be less practiced at the very manual intervention or critical-thinking called for most by such a situation. This is a studied phenomenon in human-automation interaction called an "irony of automation" and has many parallels to human-AI interaction [3]. Aside from over-reliance, pair programming may offer more than Copilot from a social perspective. Although AI is highly reliable and accurate as a programming assistant and presents a considerable advantage for computing students struggling to learn basic programming concepts, it falls short regarding its modality, which is currently limited to text and speech-to-text interaction. Human interactions, by contrast, typically involve facial expressions, body language, two-way verbal communication, etc. [48]. That is, though Copilot may match or exceed the informational capacity of peer or instructor assistance, it could fail, for other reasons, to aid the learning and academic performance of a computing student as much as a human [47].

In this study, we build upon recent work [76, 51, 59, 25] by comparing human-human pair

programming to programming with GitHub Copilot through the lens of the Control-Value theory of emotion. We conducted a mixed-methods study comparing performance, learning, workload, and emotion between these paradigms using a within-subjects design involving two study sessions one week apart. We found that despite lower workload and higher performance with GitHub Copilot, participants learned comparably and experienced more intense, positive emotions with their human partners. Our findings suggest the unique advantages of human-human programming over AI-assisted programming in computing education. We are guided by the following research questions:

- **RQ1:** To what extent does actual and perceived **programming performance** differ between the human-human and human-AI team paradigms during a learning effort?
- **RQ2:** To what extent does **learning retention** from a programming effort differ between the human-human and human-AI team paradigms?
- **RQ3:** To what extent does the **workload** (i.e., mental, temporal, effort, frustration) associated with educational programming differ between the human-human and human-AI team paradigms?
- **RQ4:** To what extent does the **emotional impact** of educational programming as measured by emotional valence and arousal differ between the human-human and human-AI team paradigms?

7.2 Related Work

7.2.1 Code-generating AI in Computing Education

Empirical research about the educational impacts of code-generating AI assistants like GitHub Copilot is mixed. Though a number of tools [46, 50, 22] have been developed specifically for use in computing education, here we focus on the broader landscape of commodity GenAI such as Copilot and ChatGPT. AI assistants are frequently observed to enhance novices' programming

performance [25, 29, 44], but have yet to show similar effects on learning [44, 42, 80]. When computing students are allowed to use AI, they sometimes use it adaptively as a learning resource but other times use it to avoid active engagement with the material [2, 53, 61, 60, 45]. Avoidant behaviors of this nature are associated with worse learning in computing courses [42] and worse psychological well-being more generally [24]. Though AI can reduce workload [34, 29] and programming anxiety [25], it may lack the conflict inherent within effective Socratic learning [6, 41]. Limited research about learning suggests that AI assistance is probably best used conservatively, in combination with traditional solo and paired paradigms [45, 51]. A possible avenue for enhancing learning would be increased motivation or self-efficacy, which tends to predict academic success [36]. One study of 45 students found that using AI in a computing course significantly improved self-efficacy [82], and at least two studies have found benefits for motivation [82, 25]. On the other hand, another study found that early and frequent AI use in a computing course was most common among students with low self-efficacy and course grades [53]. Though one study found that programmers take credit for AI's successes and blame it for failures [76], multiple others found the opposite [29, 38, 54, 79]. Overall, it seems that use of commodity AI in computing education brings much needed help at the cost of autonomy. Code-generating AI substantially changes the programming process [5], leading students to lose metacognitive control and awareness of appropriate behaviors [68, 60].

7.2.2 Pair Programming in Computing Education

Pair programming is highly regarded in computing education for its positive effects on student performance and well-being. Multiple meta-analyses have found that programming in pairs allows novices to work faster and tackle more challenging problems than they could alone, with lower or comparable effort [74, 33, 35]. Beck et al. argued for the productivity of pair programming for professionals, too, encouraging it as a tenet of the Extreme Programming (XP) method to "write all production programs with two people sitting at one machine" [7]. Pairing can even be more productive than a divide-and-conquer approach because of collaborative synergy. In

the context of education, pair programming is generally more enjoyable and better for learning than solo programming [74, 32]. Students also tend to believe in its merits, reinforcing them by a placebo-like effect [32, 19]. Challenges with educational pair programming usually stem from issues of partner compatibility (either expertise or personality) and logistical concerns. Though it is typically best to pair students of similar ability level, diversity of perspectives may stimulate more productive discussion [32, 66].

7.2.3 Learning in Human-Human v. Human-AI Pairs

To reconcile potential benefits of AI and human teammates, several studies have compared the two paradigms experimentally [48, 76, 51, 59, 25]. In 2021, before modern GenAI, Kuttal et al. compared a human tutor to a simulated automated agent scripted and performed by a human in the wizard-of-oz style [48]. They found that multi-modal communication and more lengthy, in-depth discussions were advantages of the human tutor. A later study interviewed software engineers and programming students after they programmed alone, in pairs, and with AI (ChatGPT and Copilot) [76]. They found participants were motivated by the alleviation of mundane tasks and satisfied by AI assistance, but preferred human teammates for their better knowledge, energy, and scrutiny.

Moving into education, Penney et al. conducted a between-subjects qualitative experiment of 20 CS1 students who studied a concept through text chat with either an AI tool or a human tutor [59]. The students generally found human assistants more trustworthy and reliable for learning but liked AI assistants better for fast, low-stakes, judgment-free exploration. The authors argue that novices see AI help-seeking as low-risk and low-reward, while they see human tutoring as high-risk and high-reward. Lyu et al. conducted a within-subjects study in an advanced web development course [51]. 39 students completed programming assignments in human-human, human-AI, and human-human-AI teams. Assignment scores were highest in the human-human-AI condition, followed by human-human and then human-AI. Qualitative data showed weaker partners were substantially helped by their stronger counterparts, but sometimes stronger part-

ners felt held back. Pairs of weak partners struggled to make progress together; AI helped with this, but also dominated the team in the process. Participants felt that they could communicate ineffectively and still obtain helpful results. This marks a potential harm to the metacognitive skill of analyzing and expressing a question or goal, which has previously been considered as helped by the act of prompting [61]. Finally, Fan et al. reported a quasi-experimental between-subjects study of 234 students in a web development course who programmed in either solo, human-human, or human-AI paradigms [25]. They found that performance was comparably higher in both paired conditions compared to the solo condition. Programming anxiety was most reduced by the human-AI paradigm, but perceived collaboration and social presence was highest in human-human teams.

7.2.4 Theoretical Frameworks: Control-Value and Cognitive Load

Our theoretical framework draws from Pekrun's Control-Value Theory (CVT) [58, 57] to consider the difference between human-human and human-AI programming teams through the important lens of emotional affect. CVT describes how humans' emotions follow from appraisals of both control over an achievement context and the value of achievement outcomes. When learning activities are low in value, students can feel low-energy negative emotions like boredom, for example, while feeling in-control during studying for a high value grade can induce strong positive emotions like excitement. As with related emotional theories [65, 63], CVT conceptualizes emotion with at least two basic dimensions: valence (positive v. negative) and arousal (low v. high energy). A third dimension in CVT is the object focus, which has been expanded recently beyond only the achievement activity and its outcome to also include epistemic incongruities (informational conflicts), social beings (self and others), and existential feelings of vitality (health and mortality). In the context of educational team programming, relevant foci are the activity (programming itself), the outcomes (satisfaction, grades, or money), incongruities (differences in teammates' perspectives), the self (a focus for pride or shame), and others (foci for liking or commiseration) [57]. Any of these foci can be the object of any given combination of valence

and arousal. Research in writing studies and the neuroscience of creativity likewise shows that motivation and persistence in cognitively demanding work are mediated by limbic-system activation, especially in social or emotionally meaningful contexts [27]. Human collaborators naturally stimulate the emotional and motivational circuits that sustain effort (see also [70, 69, 77, 56]), while AI tools—pleasant but affectively flat—rarely trigger the same engagement.

Considering emotional foci is useful because it broadens the possibilities for how to reach productive and enjoyable emotional states. If only considering outcome value, an AI assistant that helps complete satisfying projects or earn key grades might be chosen over a human teammate for promoting student well-being. AI assistance can also disrupt metacognition, leading students to feel overconfident in their own programming abilities on account of AI-driven progress [60]. The programming activity itself might feel more pleasant with AI for this reason. Similarly, lessened incongruities might make AI seem more emotionally preferable, since its design necessarily tries to inspire contentment, surprise, and delight over disagreement and conflict [17]. Though in isolation undesirable, conflict can be a catalyst not only for learning [42], but also for coping [63] skills that promote emotional resilience [24]. Unfortunately, sycophantic AI appeals to students' avoidant tendencies that seek to acclaim the righteousness of their existing perspectives [17]. We hypothesize that conflict, whether good or bad, is actually an essential adaptive property of human pair programming that AI is not currently effective at mimicking. Furthermore, AI's excessive competence compared to novices' is a dangerous precursor for it to assume control over programming tasks [60, 51], flattening the emotional reward of a positive outcome value according to CVT. This could also lead to a focal transition to the self. Though Bandura's theory associates high self-efficacy to low arousal, we fear that low arousal and neutral to negative valence are harmful to pride and thus self-efficacy [57]. Kallia labels this phenomenon as "AI Obscurity," whereby the credit a student normally enjoys for themselves is given instead to the AI assisting them [43]. Finally, for social foci, it is unclear how the emotional impacts of pseudo-social interactions with AI compare to those of true human social interactions. Once again, sycophantic AI creates a hazardous attraction to seek the safe agreement of a supportive AI tool, abandoning rewarding human interactions in the process [43, 17, 30].

Finally, we consider the related phenomenon of workload through the lens of Cognitive Load Theory (CLT) [73]. CLT states that learning activities have *intrinsic* difficulty as standalone tasks as well as *germane* difficulty from extracting generalizable skills and knowledge from the experience. Finally, *extraneous* load adds superfluous difficulty to the learning process due to secondary minutia such as, in the case of programming, unclear instructions or confusing interfaces. With a limited cognitive capacity, students experience that these three sources of load stack, meaning excessive extraneous load leaves too little capacity left to effectively learn from the activity. Regarding extraneous load, qualitative research is mixed regarding AI's effect for programmers. While AI offloads considerable effort for expressing the syntax of a particular problem-solving approach [64, 45, 49, 31], interacting with it adds new sources of confusion and overload [5, 61]. More importantly, there is no accepted standard for an optimal level of total workload on learning tasks, though research from disparate fields suggests that some moderate balance is best [1, 6]. Clearly, if AI removes an activity's intrinsic load, there is no stimulus to encourage growth [6]. Therefore, drawing from CVT, we anticipate human-human programming to bring together higher workload with strong positive emotions, in line with activity-based, social, and conflict-centric emotional object foci rather than an outcome focus.

7.3 Method

7.3.1 Participants

The study was conducted in accordance with the Declaration of Helsinki and approved by the Institutional Review Board (IRB) for Social and Behavioral Sciences (SBS) at the University of Virginia (IRB-SBS #7557) on July 22, 2025. Electronic informed consent was obtained from each participant. The IRB permitted us to mislead and deceive participants to support the study design, so we formally debriefed participants about this deception at the conclusion of the study. They were allowed to withdraw their consent at any time; no participants were concerned about the deception or withdrew consent.

Twenty-two participants were recruited by convenience sampling via email advertisements to roughly 900 engineering and data science students and in-class announcements to roughly 800 undergraduates across two sections of 2000-level CS (CS2; Data Structures and Algorithms) and three sections of 3000-level CS (two of Software Development and one of CS3; Data Structures and Algorithms II). Eligibility criteria included minimum age of 18, English fluency, novice or intermediate Python programming ability, and willingness to adhere to the study's multi-stage protocol. This protocol included a requirement to abstain from studying Python-related material for one week between study sessions, precluding most CS1 students from participating. Of 62 prospective participants, 25 followed through to a screening for honesty about their ability level (see screening tasks in our online appendices¹). Two were found ineligible for sub-novice ability levels, one dropped out after screening, and none were found ineligible for super-intermediate ability level. The remaining 22 participants completed the full study.

The participants were mostly male (16 male; 6 female) and Asian (15 Asian; 10 white; 1 Hispanic/Latinx) and were aged 18-29 ($\bar{x} = 20.8$; $s = 3.7$; $\tilde{x} = 18$). For ethical reasons, they were not asked where they learned about the study; however, the median age of 18 suggests that many were undergraduates in their first post-secondary year. Regarding disabilities and impairments, one participant self-reported having a physical impairment, and three reported having a cognitive or attentional impairment; none were excluded from the study. All participants had at least some experience with general purpose AI chat bots, while all but five had at least some prior experience with GitHub Copilot or a similar tool. As another measure of programming ability, we administered the Steinhorst [72] self-efficacy instrument. Summary statistics ($\bar{x} = 5.6$; $s = 0.8$; $\tilde{x} = 5.6$) indicated that participants, on average, had self-efficacy consistent with students from other institutions at the end of their introductory (CS1) courses, i.e. [72] ($\bar{x} = 5.5$; $s = 1.0$).

7.3.2 Setup

The programming setup included two Windows desktop computers, two identical 32" monitors, two keyboards, and two mice. Two versions of the Visual Studio Code (VS Code; v1.104.1) in-

¹<https://doi.org/10.5281/zenodo.19665253>

tegrated development environment (IDE) were configured on Windows 11 with Python 3.11.13 and the Pytest testing framework. Both had the standard Python extension, but one also had the GitHub Copilot (v1.372.0) extension with Chat (v0.31.1). The Chat extension was configured to use GPT 4.1—the flagship unlimited model included with Copilot’s educational free tier at the time. The non-Copilot IDE had standard Python IntelliSense auto-complete from the Python extension. Two experimental workspace folders¹ and one training folder contained sets of HumanEval tasks consisting of one PDF prompt, one Python file with an empty function to implement, and one testing file. The IDE was set up to run and test code either graphically or in the terminal. Participants were permitted to use the Google Chrome web browser and the Google search engine. AI search overviews were disabled, and using any AI in the browser was not allowed. Browsing history was tracked to ensure adherence to this restriction.

7.3.3 Tasks

Tasks were selected from a subset of the HumanEval dataset² [16] used for a prior study [29]. These hand-written Python tasks are often used to benchmark LLMs and AI code generators [81, 55]; each task comes with an English prompt and a set of automated tests. The prior study involved CS1 students completing timed HumanEval tasks either alone or with the assistance of Copilot. Eight tasks, each with a completion time of roughly 5 minutes, were organized into two pools of four tasks as similar as possible in difficulty to match the planned 20-minute time limit. The HumanEval tasks used were: Pool X (14, 52, 72, 0) and Pool Y (35, 9, 24, 18). Based on the prior study, the expected completion times of the tasks should be very similar across the two task pools. In the prior study, students completed the tasks in approximately 10 minutes on average with AI assistance, and in approximately 25 minutes solo. As mentioned in Section 7.3.2, task prompts were encoded in a PDF format. The skeleton code was stripped of the prompt text so that Copilot could not ingest it automatically. Instead, any context beyond the function stub had to be provided by the participant. Copilot was known to be very reliable for the tasks at hand, as this is the case for many simple educational programming tasks [26]. We confirmed that Copilot

²<https://github.com/openai/human-eval>

Chat was able to generate a passing solution for each of the eight tasks in one shot by copying the text of the PDF into the chat and asking Copilot to solve the problem. Participants were not advised of this or any other strategy, but were aware that Copilot could not read the contents of the PDF on its own.

7.3.4 Experimental Design

The study used a within-subjects design with the within-subjects factor of team type (Human-Human v. Human-AI). For the first study session, two participants worked together once and separately with Copilot once. Roughly one week later, they each repeated both trials separately (without Copilot). Figure 7.1 shows the eight design possibilities that were counterbalanced (rotated) across participants.

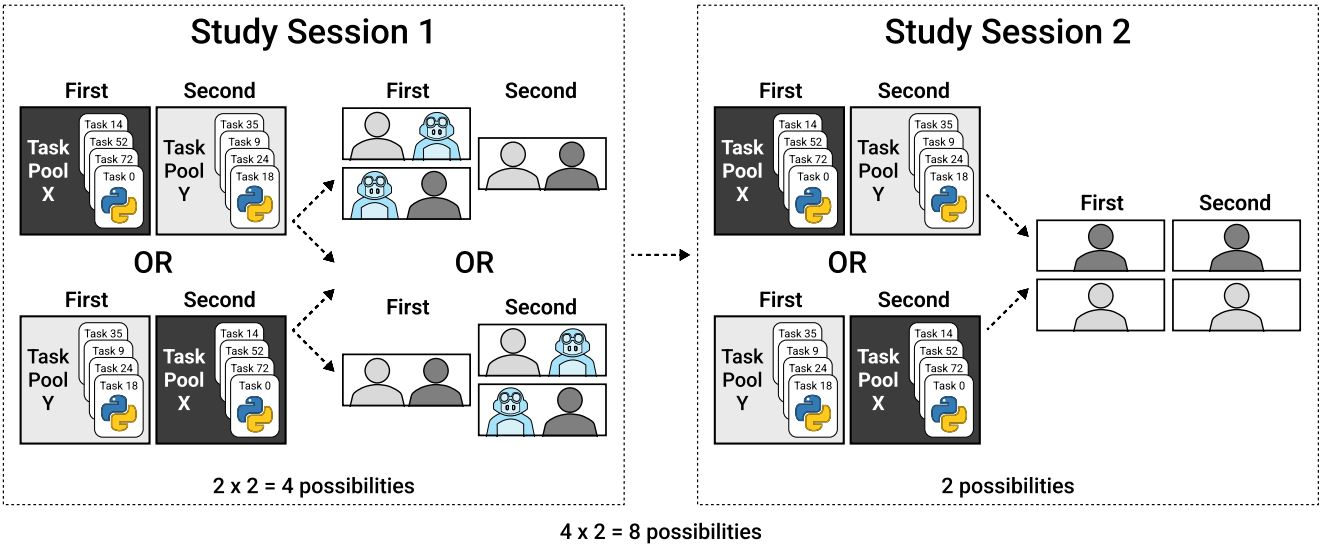


Figure 7.1: Experimental design and counterbalancing permutations.

7.3.5 Procedure

As a prerequisite for study eligibility, participants were required to attend a free one-hour workshop about Github Copilot, its benefits and drawbacks, and its major features. Next, they had to pass a screening to demonstrate their attention during the workshop and confirm their ability level

as novice or intermediate by attempting the screening tasks (see our online appendices¹) using a think-aloud technique in the presence of an experimenter. Eligible participants were paired based on their schedules and the requirement to have no prior relationships between programming partners.

On the day of their first session, a pair of participants met in the lab. They provided electronic informed consent and watched an orientation and training video³. Their assigned goal was to "balance productivity and understanding the code" as they programmed. They were deceived to believe that a final score would be calculated by a secret weighting between their Session 1 score (productivity) and their Session 2 "retention test" (understanding) score (see Equation (7.1)). They were also deceived to believe that they needed a top 15% score to win a \$10 bonus on top of their base compensation. In reality, there was no grand score calculated and all participants were given the bonus. Furthermore, information about the nature of the "retention test" was withheld from the participants until the second week, when they were informed that it was simply a repeat of the tasks from the prior week. Following the training video, participants demonstrated their competencies on a practice task pool consisting of the three screening tasks. They practiced running and testing code and using Copilot before demonstrating their competencies and understanding of the rules. They also completed a short background questionnaire that contained the Steinhurst self-efficacy inventory [72]. Next, they completed two 20-minute trials (one in each condition) according to one of the four possible design permutations (see Figure 7.1) along with questionnaires about emotion, workload, self-perceived performance, and their qualitative experience (see timeline in Figure 7.2a). Given they were balancing performance and understanding, they were instructed to use any extra time to study their solutions with their teammate, i.e., if they completed all four tasks in less than 20 minutes. Prior to the human-human trial, they were briefed on the popular driver and navigator pair programming model. They were not required to use this model and were allowed time to discuss their teamwork strategy before the timer started. At the end of each trial, code was saved in its final state for post-hoc scoring against the unit tests from HumanEval.

³<https://youtu.be/wkvKRD97ZKo>

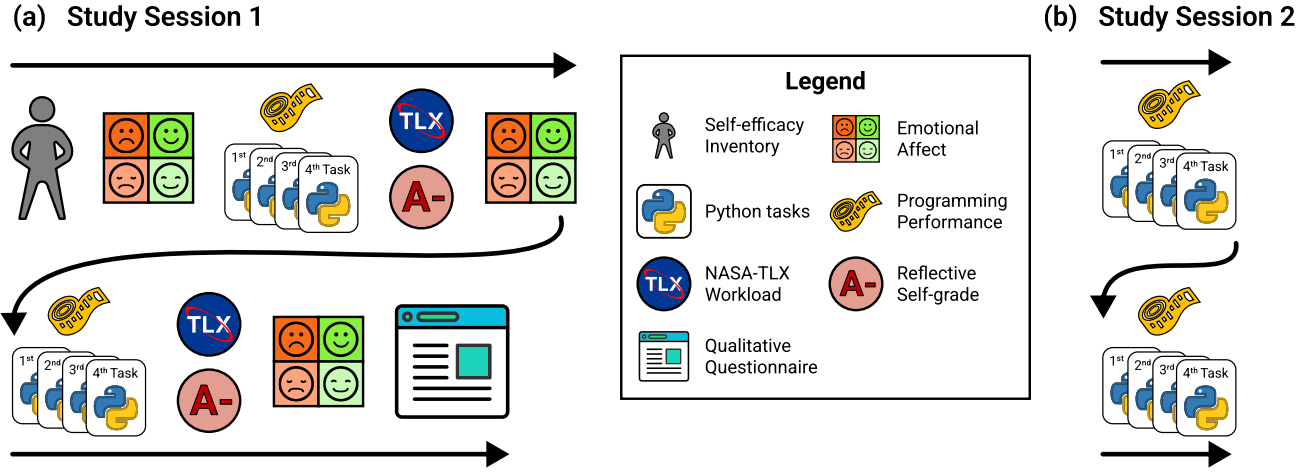


Figure 7.2: Experimental procedure.

Note. From left to right, (a) shows the procedure for study Session 1, the first exposure to the programming tasks (in Human-Human or Human-AI teams), and (b) shows the procedure for study Session 2, the second exposure to the programming tasks (solo retention test).

Following a roughly one week gap that varied slightly based on scheduling availability ($\bar{x} = \tilde{x} = 1.0$, $x_{\min} = 0.8$, $x_{\max} = 1.29$), participants returned for Session 2 where they took the retention test consisting of repeating each of the task pools from the prior week alone (see Figure 7.2b). Before leaving, participants were formally debriefed about the deception involved in the study in accordance with the approved IRB protocol. Compensation of \$7 per half hour plus a flat \$10 bonus was sent to each participant as a digital gift card.

$$\text{Score}_{\text{total}} = (\text{Score}_{\text{team}}) \cdot (?) + (\text{Score}_{\text{solo retest}}) \cdot (1 - ?) \quad (7.1)$$

$$\text{Score}_{\text{trial}} = \frac{20 \text{ pts}}{\text{task}} + \frac{5 \text{ pts}}{\text{task}} \cdot (\% \text{ time to spare}) \quad (7.2)$$

7.3.6 Data Collection

Emotion

Participants reported their subjective emotional valence and arousal three times during Session 1 (see Figure 7.2a) according to Russell's [65] circumplex model of affect. They used seven-point Likert scales (coded 1-7). To account for emotional interference between trials, the dependent measures were changes in valence and arousal ratings (after - before).

Workload

A digital version of the popular six-item NASA Task Load Index (NASA-TLX) was used to measure workload, the toll that performance takes on a human [34]. Since TLX asks about workload over a period of time, it is not represented as a difference from a baseline like emotion.

Performance and Learning

Session 1 team performance was quantified on a 100-point scale by a function (see Equation (7.2)) that combines the number of tasks completed (out of four, worth 20 points for each task) with the amount of time remaining (out of the 20 minutes allocated) at the moment that the task count was first reached. Learning was quantified via Session 2 solo performance scores using the same formula. This approach provides internally valid ratio scaling properties; i.e. completing more tasks and completing them more quickly are rewarded proportionally, and completing fewer tasks very fast can never be rewarded more than completing more tasks (e.g., completing two tasks instantly would score 50, whereas completing three tasks with no time to spare would score 60).

Perceived Performance

We also collected a self-grade from participants, as we were not confident the NASA-TLX performance dimension distinguished clearly between one's own performance and that of their team-

mate.

Qualitative Data

At the very end of Session 1, participants were asked to type free-response text comparing and contrasting the two teaming conditions and describing in what ways each was advantageous over the other. The exact questions can be found in our online appendices¹.

7.3.7 Data Analysis

All analyses were conducted in R [39] with a significance threshold of $\alpha = .05$. To estimate the effect of the treatment (AI teammate) relative to the control (Human teammate), we used Ordinary Least Squares (OLS) regression with Studentized Wild Cluster Bootstrapping (WCB) for inference [78, 52, 14]. WCB was chosen to address the small sample size and dependence in the data, including repeated observations per participant and nesting within human pairs (i.e., each participant belongs to exactly one team, so observations from teammates are correlated). Compared to alternatives, this approach provides reliable inference under multi-way clustering with few clusters [15, 78].

We tested for potential confounds due to task order or difference in the two task pools by performing Wald tests using `lmtest::waldtest(vcov = sandwich::vcovBS(R = 999, type = "wild-webb"))` [84, 85, 83]. These factors did not significantly improve model fit ($p > .1$ in all cases), indicating that the study design and counterbalancing was effective; therefore, single-predictor models were used for all research questions.

Final inference was performed using `fwildclusterboot` using `boottest(B = 9999, type = "webb")`. Effect sizes (Hedges' g) were computed and interpreted [18] with `effectsize::rm_d(method = "av", adjust = TRUE)` [9], which used mean aggregation at the team level and the Cumming [20] standardization method. Because we performed many hypothesis tests, we grouped p -values into families by research question (RQ1, RQ2, RQ3, and RQ4) and adjusted them with the Benjamini-Hochberg ("BH"; "fdr") method to control the family-wise error rate (i.e.

false discovery rate) at $\alpha = .05$ [10].

Finally, qualitative data was used to contextualize and explain each family of results. Rather than constituting a substantive standalone qualitative analysis, this was intended to help triangulate and validate the quantitative findings. As such, the method was very simple and involved one author selecting related quotes from the exit questionnaire responses to represent a diversity of perspectives from participants. To ensure fair representation of participants' perspectives, each participant was quoted exactly once, with selections distributed approximately uniformly across the four research questions. All collaborators read all exit questionnaire responses to confirm that the reported quotes fairly represented expressed sentiments. No formal coding process was performed.

7.4 Results

7.4.1 Team Performance

Figure 7.3a shows a scatterplot of the actual and perceived performance of human-human and human-AI teams during Session 1, where participants were balancing performance and understanding of their solutions. There is a clear trend for higher scores in the AI condition, with most teams completing all tasks in both conditions but faster with AI. Participants' individual self-grades, which were clearly distinguished from the team's joint performance, still generally followed the teams' actual performance. The family of hypothesis tests for RQ1 is presented in Table 7.1; the family-wise error rate was controlled between these two tests. The performance score was roughly 14 points higher out of 100 when the teammate was Copilot instead of a human, a statistically significant difference ($p_{adj} < .001$). This large [18] effect corresponds to an additional $\frac{7}{10}$ of a task or the same number of tasks but faster by a margin exceeding half of the total time limit. The upper plot in Figure 7.3b shows that both the score distribution and the median score were higher with AI. However, participants' self-grades showed a non-significant ($p > .1$) trend in the opposite direction. In other words, though participants achieved significantly better

actual performance with the AI, they did not feel that their individual performance was significantly better in either condition.

Participants could tell Copilot gave them a performance advantage, with k17 noting that "AI definitely gets things done a lot faster than humans." They also appreciated its wealth of knowledge; k4 recalled that "there was some Python stuff that both my teammate and I didn't know, but AI could provide that information." This led to conclusions that "if you are trying to get work done, the AI is much more successful" (k6) and usually "goes for an optimal solution" (k22), whereas "humans are wrong much more often" (k6). On the other hand, participants' descriptions of how they collaborated with their teammates helps to explain why they did not take credit for AI enhancement in their self-grades. Participant k12 described a multi-stage process they used with their human teammate, starting with strategies and then comparing alternative options and discussing them prior to implementation. However, with Copilot, they "didn't even think about the task" but "instead, I just copy-and-pasted the PDF and send it to AI." Being more engaged in the human-human condition made k13 feel "like I contributed more to the group." Participant k11 found this communicative process "more stimulating" than messaging Copilot "because you get to discuss what methods you think work and what methods you think don't work." In other words, there is a validation of each teammate's perspective as useful to the team. With Copilot, its superior performance makes the programmer feel as if they are much less needed.

7.4.2 Individual Learning

Participants knew they would return one week after Session 1 for a "retention test," i.e., Session 2. Unbeknownst to them, this test consisted of repeating both sets of tasks from the prior session but with no teammate of any kind. This way, their retest performance was a proxy for their learning or understanding of the tasks depending, theoretically, on with which teammate they were first exposed to that particular set of tasks. The lower plot in Figure 7.3(b) shows the distribution of retest scores by previous teammate. Learning with a human showed a trend for higher retest performance. Table 7.2 shows the family of hypothesis tests for RQ2; the family-

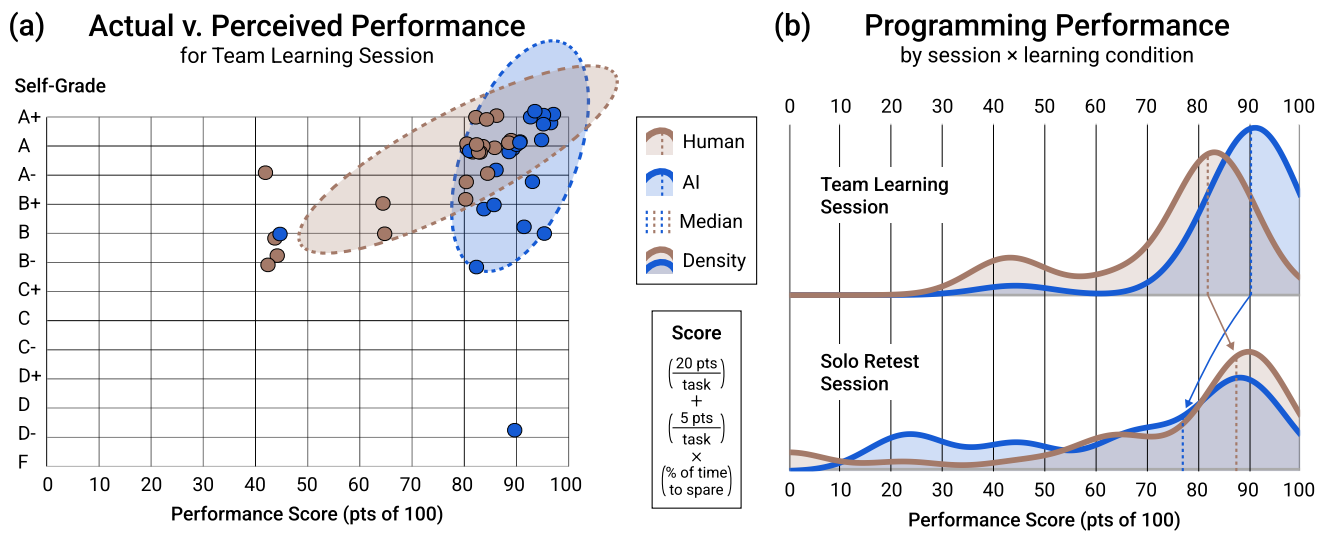


Figure 7.3: Performance plots.

Note. From left to right, (a) shows a scatter plot of actual Session 1 team performance score and self-grade for one's own contribution to Session 1 team efforts and (b) shows density plots of programming performance for team and retest trials by experimental condition.

Table 7.1: Regression coefficients of treatment on performance.

Variable	<i>B</i>	<i>CRSE</i>	95 % CI		<i>p</i>	<i>p_{adj}</i>	<i>g</i>
			<i>LL</i>	<i>UL</i>			
<i>Actual Team Performance (Session 1)</i>							
Constant	74.30	5.08	62.67	84.05			
Teammate = AI	14.09	3.96	6.46	22.89	<.001	<.001	.99
<i>Self-Grade for Team Performance (Session 1)</i>							
Constant	11.18 (A-)	0.43	10.20 (B+)	12.06 (A)			
Teammate = AI	-0.23	0.27	-0.80	0.34	.435	.435	-0.13

Note. Bolded rows are statistically significant at $\alpha = .05$. CRSE stands for Cluster Robust Standard Error (Wild Cluster Bootstrapped), p_{adj} for Benjimimi-Hochberg [10] corrected p-value for multiple-testing, and *g* is Hedges' *g* (effect size)..

wise error rate was controlled across these four tests. Relative retest performance was worse for the AI condition, a marginally significant difference ($p < .05 < p_{adj} < .1$) but absolute retest performance was not significantly different ($p > .1$). The relative constant indicates participants generally performed worse on the retest than they did working in teams, but the zero-inclusive confidence interval means their performance was statistically comparable. However, due to the higher initial performance with AI, the further retest drop by roughly 19 points of 100 in the AI condition constituted a large [18] effect (see also Figure 7.3b).

When considering only the 11 participants indicated as the weaker teammate within each of their human-human teams, the lower mean retest performance was also not statistically significant ($p > .1$). However, when considering only the 11 stronger teammates, the lower retest performance was marginally significant (after p -value adjustment; $p < .05 < p_{adj} < .1$). Hedges' g indicated this was a small [18] sized effect, at less than four points of 100 on average. The confidence intervals correspond to learning with AI costing stronger teammates between 30 seconds and 7 minutes for an equivalent score on the 20-minute retest compared to learning with the weaker teammate. Notably, though the mean absolute performance difference was larger for weaker teammates, the variability was much higher, with some retesting better on AI learned tasks and others on human learned tasks.

Participants' reflections were also mixed regarding whether the AI or human teammate was better for learning, though they generally seemed to favor the human. Because of its wide knowledge base, Copilot was fast and reliable for helping participants to understand code. Participant k1 found that Copilot "can answer more of your questions than a human can." It was also great for understanding the syntax of Python, which was helpful for many participants who were recruited from Java courses. For example, k15 said, "I felt confident in my ability to do all of the programs in Java, but not so in Python, so using Copilot felt like using training wheels on a bicycle to enhance my initial understanding of Python." However, k15 also found that Copilot often removed the need to think, whereas with the human, "we had to think out our solutions, exercising our brains." Participant k7 echoed the same sentiment, saying, "I think that the AI teammate limited me in my ability to think, whereas that was not the case with the human teammate." This pattern

also led k8 to the conclusion that "there was a better understanding of the code that was written between both people." Even though Copilot was probably a better quality learning resource than most human teammates were to each other, the natural ways that they chose to approach the interaction meant that they felt less engaged with the programming concepts in the AI condition.

Table 7.2: Regression coefficients of treatment on retest performance (learning).

Variable	<i>B</i>	<i>CRSE</i>	95 % CI		<i>p</i>	<i>p_{adj}</i>	<i>g</i>
			<i>LL</i>	<i>UL</i>			
<i>Retest Performance Change (Session 2 - Session 1)</i>							
Constant	-2.65	6.03	-15.74	10.34			
Session 1 Teammate = AI	-18.88	6.33	-32.87	-5.01	.015	.054	-1.13
<i>Absolute Retest Performance (Session 2)</i>							
Constant	71.65	5.58	59.40	83.81			
Session 1 Teammate = AI	-4.79	5.58	-16.86	7.46	.397	.529	-0.27
<i>Absolute Retest Performance for Weaker Teammates Only</i>							
Constant	55.01	10.42	31.87	77.38			
Session 1 Teammate = AI	-6.18	11.52	-30.46	18.96	.606	.606	-0.20
<i>Absolute Retest Performance for Stronger Teammates Only</i>							
Constant	88.29	2.74	82.35	92.39			
Session 1 Teammate = AI	-3.40	1.57	-6.88	-0.40	.027	.054	-0.28

Note. Bolded rows are statistically significant at $\alpha = .05$. CRSE stands for Cluster Robust Standard Error (Wild Cluster Bootstrapped), *p_{adj}* for Benjimimi-Hochberg [10] corrected p-value for multiple-testing, and *g* is Hedges' *g* (effect size).

7.4.3 Workload

To compare workload between the two teaming conditions, we explored four of the six dimensions of the NASA TLX: mental demand, temporal demand, effort, and frustration. We excluded physical demand and perceived overall performance as unhelpful for answering RQ3. Figure 7.4a shows how workload ratings following the team trials differed by condition, with workload generally appearing lower with AI. Table 7.3 shows the family of hypothesis tests for RQ3; the family-

wise error rate was controlled across these four tests. Mental demand, temporal demand, and effort were all significantly lower with the AI teammate ($p_{adj} < .01$), with large effect sizes [18]. Only frustration, which was low in both conditions, was lower by a non-significant margin ($p > .1$).

Table 7.3: Regression coefficients of treatment on subjective workload.

Variable	<i>B</i>	<i>CRSE</i>	95 % CI		<i>p</i>	<i>p_{adj}</i>	<i>g</i>
			<i>LL</i>	<i>UL</i>			
<i>TLX Mental Demand</i>							
Constant	42.95	5.74	30.52	55.59			
Teammate = AI	-23.41	5.75	-36.42	-11.77	.002	.003	-1.22
<i>TLX Temporal Demand</i>							
Constant	50.23	7.17	34.54	65.97			
Teammate = AI	-29.09	6.63	-43.68	-14.53	.001	.002	-1.21
<i>TLX Effort</i>							
Constant	44.77	4.89	34.03	55.68			
Teammate = AI	-22.95	4.35	-32.85	-13.37	.001	.002	-1.28
<i>TLX Frustration</i>							
Constant	17.05	4.55	7.16	27.14			
Teammate = AI	-3.64	4.84	-14.57	6.58	.470	.470	-0.24

Note. Four different regression models were fitted, with uncertainty quantified by refined Wild Cluster Bootstrapping. Note that the TLX Physical and TLX Performance dimensions were excluded due to lack of relevance to our research questions. Self-grade is a better measure of self-perceived performance as discussed in Section 7.3.6. Correcting p -values for multiple testing would not change the conclusions here, even with the very conservative Bonferroni adjustment. CRSE stands for Cluster Robust Standard Error (Wild Cluster Bootstrapped), p_{adj} for Benjimimi-Hochberg [10] corrected p -value for multiple-testing, and g is Hedges' g (effect size).

The feedback from the exit questionnaire aligned with the lower workload ratings observed in the AI condition. Participants mentioned how working with Copilot was "leaps and bounds easier" (k18), such that some participants were "not stressed at all" (k21). In general, Copilot's high performance was often mentioned along side a less effortful experience. Building on prior comments about thinking less with Copilot, k9 emphasized how with the human, "you have to use more brainpower." Regarding frustration, which is an emotionally charged aspect of workload, participants did not seem to mention such feelings in either case. This is consistent with the low

ratings for both conditions. Participant k3 did mention one circumstance that might be a source of frustration with Copilot, saying "It cannot follow complex logic or understand something that is not on the screen or given to it." In other words, Copilot lacked the "bigger picture" (k17) of implicit needs that humans often assume among each other. Participant k10 mentioned an emotionally salient reason for higher stress in the human condition, instead: "Programming with a human was slightly more stressful and rushed for me because I felt pressure to keep up with the pace at which my teammate was reading through the prompts and coming up with answers." On the other hand, they felt no stress with Copilot because the "fear of human judgment was removed" (k10). Perhaps it is not frustration, but rather temporal demand or some other dimension, which captures this negative experience. Though other results have suggested participants moved through content hastily with Copilot, peer pressure could have also caused weaker teammates to feign understanding before they were really ready to advance, harming their later retest performance.

7.4.4 Emotion

Emotion ratings were taken three times during Session 1 surrounding team programming. Figure 7.4b shows that ratings generally reflected positive and aroused moods. Participants were more neutrally aroused at their first measurement, with more positive and aroused emotions after programming with a human and less positive emotions after programming with AI. Figure 7.4c shows the emotional effects of the programming trials by condition. Note that depending on which ordering participants were assigned, their post-trial mood might have followed the baseline or the opposite condition. The choice of statistical method should account for this dependence. The family of hypothesis tests in Table 7.4 for RQ4 show that the emotional impact of the AI teammate was significantly less positive and less arousing than that of the human teammate; the family-wise error rate was controlled across these two tests. Confidence intervals indicate that this difference was not just a lesser positive effect, but likely a difference in sign in both cases. That is, an AI teammate seems to reduce emotional valence and arousal slightly, while a human teammate seems to increase them both slightly. These large effects [18] arise in a circumstance

where participants can easily compare the two conditions. That is, one's mood decreases with AI can be related to following a higher state from the human condition, and vice versa.

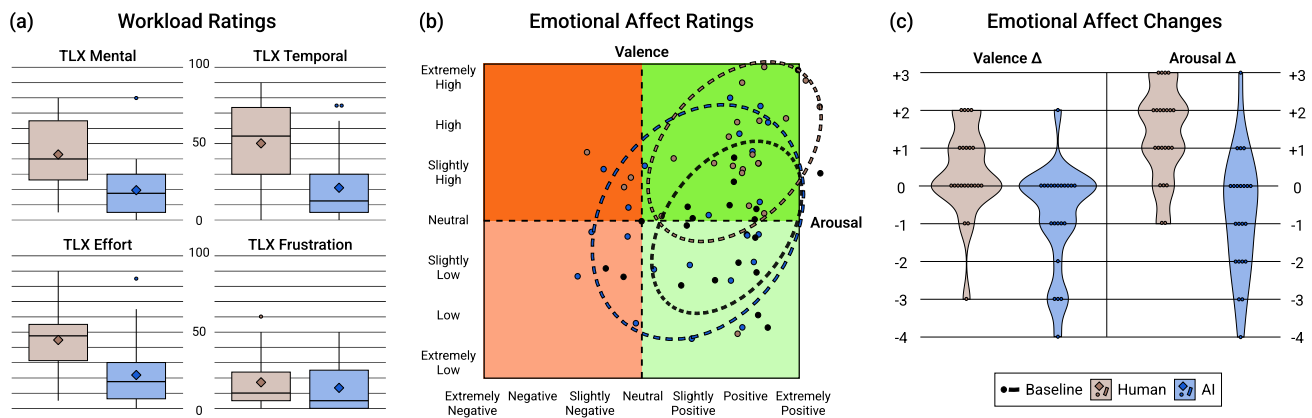


Figure 7.4: Workload and emotion plots.

Note. From left to right, (a) shows box plots of four workload sub-scales from the six-item NASA Task Load Index, excluding the physical and performance dimensions, (b) shows a scatterplot of emotional (valence, arousal) pairs taken at three time points during Session 1, the team learning session, and (c) shows violin plots of the emotional impacts (changes) of programming (during Session 1) in the two experimental conditions.

Qualitative data strongly supported the quantitative findings about emotion. As mentioned with the frustration dimension of workload, there does seem to be some crossover between mood and workload. For example, k19 reported that "emotionally speaking, working with a human partner will give you different sense of courage and stress." That is, the human gives you the confidence to work through challenges, but also raises the stakes, heightening emotional arousal and perhaps workload. In terms of valence, participants like k5 found that "working with a teammate felt better because you help each other which gives more fulfillment." Similarly, k14 said that with the human, "I feel like my overall mood improved." Finally, one comment crystallizes how some participants likely struggled to disentangle valence from arousal when providing their ratings. According to k16, working with a human "keeps the mental and social energy up, so I was more happy/excited when coding." They experienced feeling not only "energized" (arousal) but also "happy" (valence) and "excited" (valence and arousal). Another interpretation is that valence and arousal simply tended to move together.

Table 7.4: Regression coefficients of treatment on emotion.

Variable	<i>B</i>	<i>CRSE</i>	95 % CI		<i>p</i>	<i>p_{adj}</i>	<i>g</i>
			<i>LL</i>	<i>UL</i>			
<i>Emotional Valence Change</i>							
Constant	0.36	0.14	0.07	0.64			
Teammate = AI	-1.18	0.31	-1.84	-0.52	.001	.002	-1.61
<i>Emotional Arousal Change</i>							
Constant	1.36	0.27	0.77	1.95			
Teammate = AI	-2.09	0.58	-3.37	-0.83	.004	.004	-1.74

Note. Dependent measures were rating changes over a trial (after - before) to represent the emotional impact of the trial. CRSE stands for Cluster Robust Standard Error (Wild Cluster Bootstrapped), *p_{adj}* for Benjimimi-Hochberg [10] corrected p-value for multiple-testing, and *g* is Hedges' *g* (effect size).

7.5 Discussion

7.5.1 RQ1: To what extent does actual and perceived programming performance differ between the human-human and human-AI team paradigms during a learning effort?

Overall, we found that participants performed better with Copilot than in pairs, but this was not reflected by correspondingly high self-grades. Participants did not feel worse about their own performance when using Copilot; they simply did not take personal credit for its help. Qualitative data painted a broader picture, showing that participants felt more useful and valued with human partners, though this outcome wasn't fully reflected in their self-grades. One likely reason for the mismatch between these confidence reports is that Copilot's consistently superior performance made students reluctant to credit themselves for success. Copilot's superior knowledge base and task performance positioned it as the unequivocally stronger teammate, and because its responses were both confident and sycophantic, participants often experienced it as taking over the problem and supplying a perfect solution, leaving little space to see their own contribution. From

a Control-Value Theory perspective [58], working with Copilot clearly increased outcome value but reduced the programmer's control over that outcome. Copilot's assistance also reduced activity focal value by taking away the meaningful challenge associated with programming. In the human team, however, leadership was negotiable and shifting, and shared goals were assumed. Participants knew their teammate was of similar ability and pursuing the same balance between performance and understanding. By contrast, few participants took the time to articulate contextual factors like these to Copilot, contributing to participants finding Copilot to be less collaborative than their human teammate. CVT predicts social object foci (oneself or a compatriot) to evoke positive emotions when either partner can contribute to a shared goal [57]. For example, being helpful to another inspires satisfaction and pride, while being helped by another evokes gratitude and liking. If the balance of these were to shift, as when partners differ greatly in ability level [32] or Copilot outshines the programmer, value can be high, but control is low. That is, a weak partner becomes reliant on the stronger (or Copilot) to obtain either outcome, activity, or social value. Interventions, such as requiring context specifications or enabling multi-modal communication with AI, could narrow the current gap, but for now, human interaction still supports these collaborative dynamics more effectively than generative AI.

7.5.2 RQ2: To what extent does learning retention from a programming effort differ between the human-human and human-AI team paradigms?

There was a trend toward worse learning retention for tasks learned with AI rather than with a human teammate, but these differences were mostly not significant. This general trend aligns with recent empirical work showing that AI assistance during learning can impair long-term knowledge retention [4]. Curiously, participants' absolute retest performance actually aligned better to their initial self-grades than to their team performance, suggesting that students may be better judges of their own learning than objective assessments corrupted by AI involvement. Relative to the initially higher performance with AI compared to the human, there was a larger performance drop (marginally significant) for the retest when repeating tasks learned with AI. It might be expected

from the theories of Zone of Proximal Development (ZPD; [77]) or Flow (ZPF; [6, 29]), that weaker teammates would learn better by their teammates' pushing them slightly beyond their current ability. Yet, it was instead stronger teammates who had slightly lower (marginally significant) absolute retest performance for the AI tasks. Aside from random chance, one explanation for this is that stronger teammates were acting as teachers in the interaction. Since they had to understand the code well enough to help their weaker counterpart understand it, too, perhaps they engaged with the code more deeply, allowing them to reproduce it more quickly in the retest. This model, called *near-peer teaching* within medical education, does seem to correlate with academic performance for the teaching student, though selection bias is likely present in observational studies [47]. Another possibility is that stronger teammates served as a "rate limiter", pacing the team according to their own understanding. As noted in Section 7.4.3, weaker teammates may have hesitated to admit confusion or request a slower pace. The confidence intervals in Table 7.2 align with this interpretation, in that weaker teammates trended toward better learning with humans, though the effect was inconsistent. While many weaker teammates may have benefited from human explanations, others may have learned more effectively with Copilot, where they could progress at their own pace. Finally, as was expected, Copilot helped participants complete tasks much more quickly thus leaving plenty of time for passive studying at the end of Session 1. In contrast, human pairs sometimes finished late or with little time to review and consolidate their understanding. This may have disadvantaged weaker teammates in particular, who may have had unresolved questions but lacked time or confidence to raise them.

7.5.3 RQ3: To what extent does the workload associated with educational programming differ between the human-human and human-AI team paradigms?

Mental demand, temporal demand, and effort were all significantly lower with Copilot, while frustration was not significantly different. These findings indicate that learning is likely more active in a human-human team, but there is a possibility of covering less material before becoming cogni-

tively fatigued [73]. The differences in mental demand and effort mirror a prior study comparing solo and AI-assisted paradigms [29]. However, that study did not find a conclusive difference in temporal demand. A possible reason for this is that there were many more tasks available than participants would likely have been able to complete in the time limit. The result was that very few participants completed all tasks with any time to spare. By comparison, the task pools here were designed to be completed approximately within the time limit, so many more participants completed them with time to spare. As such, participants here likely felt considerably less rushed in the AI condition, where they were not working up to the time limit. Frustration trended lower in the AI condition, but not significantly so, in line with prior work [31, 31, 29]. It seems that participants were not particularly frustrated in either condition. Whereas frustration is a workload state associated with high arousal negative emotion, participants generally reported positive moods throughout the study.

7.5.4 RQ4: To what extent does the emotional impact of educational programming as measured by emotional valence and arousal differ between the human-human and human-AI team paradigms?

Though mood ratings were overall positive in valence and neutral in arousal, the emotional impact of programming was significantly more positive and arousing with the human teammate compared to Copilot. This led to an overall more enjoyable and engaging experience despite lower performance and higher workload. From a Control-Value Theory perspective, these findings suggest that human collaboration increased both the activity value and social value, producing high-arousal positive emotions [57]. As seen in the qualitative emotion results, social value is the fulfillment from two-way reliance, which maintains control on both sides; the activity itself also becomes more fun and therefore more valuable. Findings also supported that productive team conflict encouraged curiosity at the epistemic incongruity object focus [57]. This is an exciting finding, suggesting that students are willing to invest more effort in the learning process when working with a human partner than with an AI coding assistant. Research on desirable difficulties

shows that moderate challenge can promote learning, whereas tasks that are too easy may reduce meaningful cognitive effort [13]. Workload alone therefore provides an incomplete picture of the learning process. While excessive workload can lead to discouragement (or frustration), this was not substantially lower with Copilot, and the more positive emotions observed in the human pairs suggest that the higher workload was productive rather than harmful. Arousal tells a similar story, with the mean difference of two Likert points out of seven pointing to a heightened alertness within the true social interaction. Given that optimal arousal is important for human performance [1, 67], it is likely that the participants were performing better as individuals in the human team even as Copilot artificially inflated their AI-assisted scores. This was also reflected by the self-grade findings discussed in Sections 7.5.1 and 7.5.2.

7.5.5 Implications & Recommendations

We strongly recommend that educators look for opportunities to integrate human-human pair programming into their programming courses. The strong emotional benefits amidst lower performance and workload suggest that students will work harder more happily with their peers than with AI. There is a clear benefit and appeal for students to use Copilot on their coursework. It will make programming easier and faster for simple problems, potentially without the major learning consequences that many fear. Going beyond the conditions of this study, however, it is easy to imagine how human-human pair programming could surpass AI-assisted programming. Most participants here had plenty of spare time to study passively with Copilot, making efficient use of a fixed, 20-minute period for that purpose. The slower and more active studying process of the human-human condition either left participants with better understanding of less material (i.e., tasks completed) or a better but incomplete understanding of a similar amount of material. What is not clear is how additional time might be used in a non-experimental setting. With Copilot, participants rarely needed additional time to further study the tasks. With the human teammate, however, participants were so energized and engaged that they might reasonably have taken extra time if they were allowed.

Similar to prior work [60, 75], our results indicate that Copilot can still be used as an effective educational tool and should not be dismissed completely. With the observed performance benefit and inconclusive learning effect, Copilot seems to offer very accessible and low-stakes information retrieval for students desiring to learn. Prior work has shown evidence that human-human teams can be augmented with AI for a best-of-both worlds scenario [51]. It seems plausible from our findings here that having Copilot or another AI-based tool to fill knowledge gaps would have been helpful to some of our human-human teams. However, the loss of control and autonomy reported in prior work [11] was also replicated here and makes introducing AI into human-human teams hazardous. While power dynamics already govern human pair programming [21], adding an omniscient oracle risks an automatic deferral away from student control. One possible solution would be to encourage pairs of students to voluntarily modify the behavior of their AI tools to align with their learning goals via intentional use of system guardrails. Another possible feature, which Copilot does not offer, is user-defined rate limiting. Perhaps if students could view Copilot’s help as more scarce, they might take better advantage of their human teammate in the average case, only turning to Copilot when stuck. Prior work has shown that students who turn to AI later in the problem-solving process tend to perform better [53]. This could also help with students being annoyed by Copilot’s proactive, sometimes too eager, autocompletion suggestions observed in prior work [61]. Overall, while there is no reason that the benefits of both paradigms cannot theoretically be combined, extra care needs to be taken to ensure that the social benefits of human interaction are not disrupted by adding Copilot into the mix— prior work has found that easy access to generative AI models is already eroding social interactions in introductory programming classes [37].

7.6 Threats to Validity

Regarding internal validity, reliable inference is very challenging with a small sample and the multi-way clustering inherent to this area of inquiry. We defend our statistical method as a best-in-class approach to handling our data but acknowledge that spurious results (Type II errors) are

possible. We used p -value adjustments within each of four families (one per RQ) to mollify risks to our conclusions. Regarding Type I error, we reported confidence intervals and effect sizes to contextualize our results. With respect to endogeneity, we designed our study with careful counterbalancing procedures in mind and made our two task pools as similar as possible. To ensure no major confounding effects remained from ordering or task pool differences, we also used Wald tests to challenge the parsimony of our regressions. A final concern is the dependence structure of emotion measures. The within-subjects design means that participants' changes in emotion were possibly influenced in magnitude based on the ordering of their trials. Counterbalancing should have mitigated any bias to the conclusion of RQ4, but the absolute magnitude of emotion changes might show higher uncertainty based on the impossibility of resetting one's emotion back to baseline. As such, the absolute changes in mood ratings should be interpreted with caution.

As far as external validity, this work was conducted with a small group of highly organized volunteers at a selective research university in the eastern United States, so attempts to generalize our findings should consider this context. The pairing of participants was also done by convenience and may not reflect an intentional pairing strategy performed in an instructional setting [32]. As discussed in Section 7.5.5, we suspect that participants might have behaved differently in the two experimental conditions had they been offered additional time in a naturalistic setting. Ultimately, real students will not spend a fixed amount of time learning regardless of whether they collaborate with a human or an AI assistant. More likely, an internal risk-reward balance will determine how much time they invest [71]. Further research should explore what factors determine how much time a student is willing to invest into educational collaborative programming when allowed to choose for themselves. Finally, it is challenging to balance design consistency with the realistic complexity of real-world programming tasks. Copilot was known to be reliable at solving the tasks selected from HumanEval, but it remains to be seen whether messier or more poorly specified tasks might have been performed or learned better through a human-human paradigm. Additionally, it is possible that for such tasks, the human might need to be more involved in helping Copilot, such that differences observed here might fade or change.

7.7 Conclusion

This mixed-methods laboratory study used data from an experiment with 11 pairs of novice and intermediate programmers to compare performance, learning, workload, and emotion between human-human and human-AI team paradigms. Prior work suggested that an AI teammate would improve objective performance and reduce workload compared to a human teammate, but it was not clear how self-perceived performance, learning, or emotion would differ. We used linear regression and Wild Cluster Bootstrapping to perform within-subjects comparisons between the two experimental conditions, which were administered in a counterbalanced design. A qualitative questionnaire helped to explain the quantitative results. We found significantly higher performance and lower mental demand, temporal demand, and effort in the AI condition but mostly inconclusive effects on self-perceived performance, frustration, and learning. There was some very limited evidence for worse learning in the AI condition, especially for participants who were the stronger member of their human-human team. The emotional effect of programming was significantly more positive and arousing in the human condition. We conclude that students will likely work harder and more happily in educational programming contexts with human teammates than with GitHub Copilot. We recommend that human-human pair programming be reconsidered for its power as an educational tool, and recommend that students maintain autonomy and social activity if they choose to augment their human-human teams with AI assistance.

Acknowledgments

We thank Jennifer Burman and the UVA Link Lab for supporting the AI Coding Workshop series where participants were trained. This work was supported by the University of Virginia Distinguished Fellowship, NSF National Research Traineeship for Cyber-Physical Systems (1829004), Graduate Research Fellowship Program for Computer and Information Science and Engineering (1842490), and the Virginia Commonwealth Cyber Initiative Central Virginia Node (GR102900).

References

- [1] Kazunori Akizuki and Yukari Ohashi. 2015. Measurement of functional task difficulty during motor learning: what level of difficulty corresponds to the optimal challenge point? *Human Movement Science*, 43, (Oct. 1, 2015), 107–117. doi:10.1016/j.humov.2015.07.007.
- [2] Matin Amoozadeh, Daye Nam, Daniel Prol, Ali Alfageeh, James Prather, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Student-AI interaction: a case study of CS1 students. (Oct. 11, 2024). arXiv: 2407.00305[cs]. doi:10.48550/arXiv.2407.00305.
- [3] Lisanne Bainbridge. 1983. Ironies of automation. In *Analysis, design and evaluation of man-machine systems*. Elsevier, 129–135.
- [4] André Barcaui. 2025. ChatGPT as a cognitive crutch: Evidence from a randomized controlled trial on knowledge retention. *Social Sciences & Humanities Open*, 12, 102287. doi:https://doi.org/10.1016/j.ssaho.2025.102287.
- [5] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: how programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7, 85–111, OOPSLA1. Number: OOPSLA1.
- [6] Ashok R. Basawapatna, Alexander Repenning, Kyu Han Koh, and Hilarie Nickerson. 2013. The zones of proximal flow: guiding students through a space of computational thinking skills and challenges. In *Proceedings of the ninth annual international ACM conference on International computing education research*. ICER '13: International Computing Education Research Conference. ACM, San Diego San California USA, (Aug. 12, 2013), 67–74. isbn: 978-1-4503-2243-0. doi:10.1145/2493394.2493404.
- [7] Kent Beck, Cynthia Andres, and O'Reilly Online Learning: Academic/Public Library Edition. 2004. *Extreme Programming Explained: Embrace Change. The XP Ser.* Addison Wesley Professional, Boston, (Nov. 16, 2004). isbn: 978-0-321-27865-4 0-321-27865-8. http://RE5QY4SB7X.search.serialssolutions.com/?V=1.0&L=RE5QY4SB7X&S=JCs&C=TC0000074886&T=marc.
- [8] 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*, 500–506.
- [9] Mattan S. Ben-Shachar, Daniel Lüdecke, and Dominique Makowski. 2020. Effectsize: estimation of effect size indices and standardized parameters. *Journal of Open Source Software*, 5, 56, 2815. doi:10.21105/joss.02815.

- [10] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57, 1, 289–300. Number: 1.
- [11] Simon Berger and Julian Voigt. 2025. My Way or the AI Way? How Autonomy Frustration Undermines Creative Performance Through Editing AI Advice. *How Autonomy Frustration Undermines Creative Performance Through Editing AI Advice (April 28, 2025)*.
- [12] 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. *Commun. ACM*, 66, 6, (May 24, 2023), 56–62. doi:10.1145/3589996.
- [13] Robert A. Bjork. 1994. Memory and Metamemory Considerations in the Training of Human Beings. In *Metacognition: Knowing About Knowing*. Janet Metcalfe and Arthur P. Shimamura, (Eds.) MIT Press, Cambridge, MA, 185–205.
- [14] A. Colin Cameron, Jonah B. Gelbach, and Douglas L. Miller. 2008. Bootstrap-based improvements for inference with clustered errors. *The review of economics and statistics*, 90, 3, 414–427. Retrieved Feb. 13, 2026 from <https://direct.mit.edu/rest/article-abstract/90/3/414/57731>.
- [15] A. Colin Cameron and Douglas L. Miller. 2015. A practitioner’s guide to cluster-robust inference. *Journal of human resources*, 50, 2, 317–372. Retrieved Feb. 13, 2026 from <https://jhr.uwpress.org/content/50/2/317.short>.
- [16] Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- [17] Myra Cheng, Cino Lee, Pranav Khadpe, Sunny Yu, Dyllan Han, and Dan Jurafsky. 2025. Sycophantic AI decreases prosocial intentions and promotes dependence. (Oct. 1, 2025). arXiv: 2510.01395[cs]. doi:10.48550/arXiv.2510.01395.
- [18] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Academic press.
- [19] Ben Colagiuri, Evan J. Livesey, and Justin A. Harris. 2011. Can expectancies produce placebo effects for implicit learning? *Psychonomic Bulletin & Review*, 18, 2, (Apr. 2011), 399–405. doi:10.3758/s13423-010-0041-1.
- [20] Geoff Cumming. 2013. *Understanding the new statistics: Effect sizes, confidence intervals, and meta-analysis*. Routledge. Retrieved Feb. 19, 2026 from <https://api.taylorfrancis.com/content/books/mono/download?identifierName=doi&identifierValue=10.4324/9780203807002&type=googlepdf>.
- [21] Elise Deitrick, R Benjamin Shapiro, and Brian Gravel. 2016. How do we assess equity in programming pairs? In Singapore: International Society of the Learning Sciences.

- [22] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt problems: a new programming exercise for the generative AI era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 296–302. isbn: 979-8-4007-0423-9. doi:10.1145/3626252.3630909.
- [23] Paul Denny et al. 2024. Computing education in the era of generative AI. *Commun. ACM*, 67, 2, (Jan. 2024), 56–67. doi:10.1145/3624720.
- [24] Maria TM Dijkstra and Astrid C. Homan. 2016. Engaging in rather than disengaging from stress: effective coping and perceived control. *Frontiers in psychology*, 7, 1415. Retrieved Feb. 20, 2026 from <https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2016.01415/full>.
- [25] Guangrui Fan, Dandan Liu, Rui Zhang, and Lihu Pan. 2025. The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, 12, 1, (Mar. 4, 2025), 16. doi:10.1186/s40594-025-00537-3.
- [26] 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* (ACE '23). Association for Computing Machinery, New York, NY, USA, (Jan. 30, 2023), 97–104. isbn: 978-1-4503-9941-8. doi:10.1145/3576123.3576134.
- [27] Alice Weaver Flaherty. 2004. *The Midnight Disease: The Drive to Write, Writer's Block, and the Creative Brain*. Houghton Mifflin, Boston. isbn: 978-0-618-23065-5.
- [28] Nat Friedman. 2021. Introducing GitHub copilot: your AI pair programmer. The GitHub Blog. (June 29, 2021). Retrieved July 27, 2023 from <https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/>.
- [29] Nicholas Gardella, Raymond Pettit, and Sara L. Riggs. 2024. Performance, workload, emotion, and self-efficacy of novice programmers using AI code generation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITICSE 2024: Innovation and Technology in Computer Science Education. ACM, Milan Italy, (July 3, 2024), 290–296. isbn: 979-8-4007-0600-4. doi:10.1145/3649217.3653615.
- [30] Nicholas Gardella, Joseph Shelton, Isabella Graßl, and Sara Riggs. 2025. HBCU student perspectives on identity, persistence, and code-generating AI in CS education: a case study. In *Proceedings of the 25th Koli Calling International Conference on Computing Education Research*. Koli Calling '25: 25th Koli Calling

- International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 11, 2025), 1–12. isbn: 979-8-4007-1599-0. doi:10.1145/3769994.3770015.
- [31] Gian Luca Scoccia. 2023. Exploring early adopters' perceptions of ChatGPT as a code generation tool, (Sept. 11, 2023). MAG ID: 4388214696. doi:10.1109/asew60602.2023.00016.
- [32] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. Pair programming in education: a literature review. *Computer Science Education*, 21, 2, 135–173.
- [33] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I. K. Sjøberg. 2009. The effectiveness of pair programming: a meta-analysis. *Information and Software Technology*. Special Section: Software Engineering for Secure Systems 51, 7, (July 1, 2009), 1110–1122. doi:10.1016/j.infsof.2009.02.001.
- [34] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (task load index): results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.
- [35] Anja Hawlitschek, Sarah Berndt, and Sandra Schulz. 2023. Empirical research on pair programming in higher education: a literature review. *Computer Science Education*, 33, 3, (July 3, 2023), 400–428. doi:10.1080/08993408.2022.2039504.
- [36] Toni Honicke and Jaclyn Broadbent. 2016. The influence of academic self-efficacy on academic performance: a systematic review. *Educational research review*, 17, 63–84. Retrieved Dec. 29, 2023 from <https://www.sciencedirect.com/science/article/pii/S1747938X15000639>.
- [37] Irene Hou, Owen Man, Kate Hamilton, Srishty Muthusekaran, Jeffin Johnykutty, Leili Zadeh, and Stephen MacNeil. 2025. 'All Roads Lead to ChatGPT': How Generative AI is Eroding Social Interactions and Student Learning Communities. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 79–85. isbn: 979-8-4007-1567-9. doi:10.1145/3724363.3729024.
- [38] Irene Hou, Sophia Mettelle, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The effects of generative AI on computing students' help-seeking preferences. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*. Association for Computing Machinery, New York, NY, USA, (Jan. 29, 2024), 39–48. isbn: 979-8-4007-1619-5. doi:10.1145/3636243.3636248.
- [39] Ross Ihaka and Robert Gentleman. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5, 3, (Sept. 1996), 299–314. doi:10.1080/10618600.1996.10474713.
- [40] Sumeet Jeswani, Akshay Mittal, and Soni Sewlani. 2025. Premature trust: how student overreliance on GenAI is seeding tomorrow's security gaps. In *Proceedings of the 26th ACM Annual Conference on Cybersecurity & Information Technology Education (SIGCITE '25)*. Association for Computing Machinery, New York, NY, USA, 107–112. isbn: 979-8-4007-2240-0. doi:10.1145/3769694.3771124.

- [41] Martin Jonsson and Jakob Tholander. 2022. Cracking the code: co-coding with AI in creative programming education. In *Proceedings of the 14th Conference on Creativity and Cognition (C&C '22)*. Association for Computing Machinery, New York, NY, USA, (June 20, 2022), 5–14. isbn: 978-1-4503-9327-0. doi:10.1145/3527927.3532801.
- [42] Gregor Jošt, Viktor Taneski, and Sašo Karakatič. 2024. The impact of large language models on programming education and student learning outcomes. *Applied Sciences*, 14, 10, (Jan. 2024), 4115. Number: 10. doi:10.3390/app14104115.
- [43] Maria Kallia. 2025. To be, or to be otherwise? silicon souls in search of dasein and authentic human engagement in the age of generative AI. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1 (ICER '25)*. Association for Computing Machinery, New York, NY, USA, (Aug. 2, 2025), 240–255. isbn: 979-8-4007-1340-8. doi:10.1145/3702652.3744223.
- [44] 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, (Apr. 19, 2023), 1–23. isbn: 978-1-4503-9421-5. doi:10.1145/3544548.3580919.
- [45] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–12. isbn: 979-8-4007-1653-9. doi:10.1145/3631802.3631806.
- [46] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, (May 11, 2024), 1–20. isbn: 979-8-4007-0330-0. doi:10.1145/3613904.3642773.
- [47] András Komócsi, Gergely Csaba, Eszter Veronika Csöngéi, Krisztina Fischer, Kristóf Filipánits, László Czopf, and Andrea Tamás. 2026. Academic performance and progression among near-peer tutors: a comparative analysis in undergraduate medical education. *Medical Teacher*, 48, 3, (Mar. 4, 2026), 394–404. _eprint: <https://doi.org/10.1080/0142159X.2025.2553658>. doi:10.1080/0142159X.2025.2553658.

- [48] Sandeep Kaur Kuttal, Bali Ong, Kate Kwasny, and Peter Robe. 2021. Trade-offs for substituting a human with an agent in a pair programming context: the good, the bad, and the ugly. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (CHI '21). Association for Computing Machinery, New York, NY, USA, (May 7, 2021), 1–20. isbn: 978-1-4503-8096-6. doi:10.1145/3411764.3445659.
- [49] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A large-scale survey on the usability of AI programming assistants: successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (ICSE '24). Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3608128.
- [50] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. Koli Calling '23: 23rd Koli Calling International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 13, 2023), 1–11. isbn: 979-8-4007-1653-9. doi:10.1145/3631802.3631830.
- [51] Wenhan Lyu, Yimeng Wang, Yifan Sun, and Yixuan Zhang. 2025. Will your next pair programming partner be human? an empirical evaluation of generative AI as a collaborative teammate in a semester-long classroom setting. In *Proceedings of the Twelfth ACM Conference on Learning @ Scale*. L@S '25: Twelfth ACM Conference on Learning @ Scale. ACM, Palermo Italy, (July 17, 2025), 83–94. isbn: 979-8-4007-1291-3. doi:10.1145/3698205.3729544.
- [52] James G. MacKinnon, Morten Ørregaard Nielsen, and Matthew D. Webb. 2023. Fast and reliable jackknife and bootstrap methods for cluster-robust inference. *Journal of Applied Econometrics*, 38, 5, (Aug. 2023), 671–694. doi:10.1002/jae.2969.
- [53] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-regulation, self-efficacy, and fear of failure interactions with how novices use LLMs to solve programming problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2024: Innovation and Technology in Computer Science Education. ACM, Milan Italy, (July 3, 2024), 276–282. isbn: 979-8-4007-0600-4. doi:10.1145/3649217.3653621.
- [54] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24: IEEE/ACM 46th International Conference on Software Engineering. ACM, Lisbon Portugal, (Apr. 12, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3639187.

- [55] Palanichamy Naveen, T. Akilan, P. Manikandan, C. Swedheetha, and M. Saravanan. 2025. Comparative review of large language models. In *2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)*. IEEE, 2120–2124. Retrieved Feb. 11, 2026 from https://ieeexplore.ieee.org/abstract/document/11170906/?casa_token=8Dcr0MItU_cAAAAA:3qj-JPYir4hqr1JDkCm4GQL6QwDEiDgQh0vGsUuV-GygRbXVv-CA3210PKE4BnAQv3yE4hE4BIE.
- [56] Annie Murphy Paul. 2021. *The Extended Mind: The Power of Thinking Outside the Brain*. Houghton Mifflin Harcourt, Boston, MA. isbn: 978-0-544-94766-5.
- [57] Reinhard Pekrun. 2024. Control-value theory: from achievement emotion to a general theory of human emotions. *Educational Psychology Review*, 36, 3, (Sept. 2024), 83. doi:10.1007/s10648-024-09909-7.
- [58] Reinhard Pekrun. 2006. The control-value theory of achievement emotions: assumptions, corollaries, and implications for educational research and practice. *Educational Psychology Review*, 18, 4, (Nov. 30, 2006), 315–341. doi:10.1007/s10648-006-9029-9.
- [59] Jacob Penney, Pawan Acharya, Peter Hilbert, Priyanka Parekh, Anita Sarma, Igor Steinmacher, and Marco Gerosa. 2025. Understanding programming students' help-seeking preferences in the era of generative AI. In *Proceedings of the ACM Global Computing Education Conference 2025 - Volume 1*. CompEd 2025: ACM Global Computing Education Conference 2025. ACM, Gaborone Botswana, (Oct. 21, 2025), 15–21. isbn: 979-8-4007-1929-5. doi:10.1145/3736181.3747165.
- [60] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: the benefits and harms of generative AI for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*. ICER 2024: ACM Conference on International Computing Education Research. ACM, Melbourne VIC Australia, (Aug. 12, 2024), 469–486. isbn: 979-8-4007-0475-8. doi:10.1145/3632620.3671116.
- [61] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. “it’s weird that it knows what i want”: usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 29, 2024), 1–31. doi:10.1145/3617367.
- [62] James Prather et al. 2025. Beyond the hype: a comprehensive review of current trends in generative AI research, teaching practices, and tools. In *2024 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 300–338. isbn: 979-8-4007-1208-1. doi:10.1145/3689187.3709614.

- [63] Ira J. Roseman. 1996. Appraisal determinants of emotions: constructing a more accurate and comprehensive theory. *Cognition & Emotion*, 10, 3, (May 1996), 241–278. doi:10.1080/026999396380240.
- [64] 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. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. IUI '23: 28th International Conference on Intelligent User Interfaces. ACM, Sydney NSW Australia, (Mar. 27, 2023), 491–514. isbn: 979-8-4007-0106-1. doi:10.1145/3581641.3584037.
- [65] James A. Russell. 1980. A circumplex model of affect. *Journal of personality and social psychology*, 39, 6, 1161. Retrieved Nov. 7, 2023 from <https://psycnet.apa.org/journals/psp/39/6/1161/>.
- [66] Norsaremah Salleh, Emilia Mendes, and John Grundy. 2011. Empirical studies of pair programming for CS/SE teaching in higher education: a systematic literature review. *IEEE Transactions on Software Engineering*, 37, 4, (July 2011), 509–525. doi:10.1109/TSE.2010.59.
- [67] Benjamin T. Sharpe and Ian Tyndall. 2025. The sustained attention paradox: a critical commentary on the theoretical impossibility of perfect vigilance. *Cognitive Science*, 49, 4, e70061. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cogs.70061>. doi:10.1111/cogs.70061.
- [68] Abdulhadi Shoufan. 2023. Can students without prior knowledge use ChatGPT to answer test questions? an empirical study. *ACM Trans. Comput. Educ.*, 23, 4, (Dec. 11, 2023), 45:1–45:29. doi:10.1145/3628162.
- [69] Nancy Sommers. 1982. Responding to student writing. *College Composition & Communication*, 33, 2, 148–156. Type: Journal Article. doi:<https://doi.org/10.58680/coc198215854>.
- [70] Nancy Sommers. 1980. Revision strategies of student writers and experienced adult writers. *College Composition & Communication*, 31, 4, 378–388. Type: Journal Article. doi:<https://doi.org/10.58680/coc198015930>.
- [71] Lisa K. Son and Rajiv Sethi. 2006. Metacognitive control and optimal learning. *Cognitive Science*, 30, 4, (July 8, 2006), 759–774. doi:10.1207/s15516709cog0000_74.
- [72] Phil Steinhorst, Andrew Petersen, and Jan Vahrenhold. 2020. Revisiting self-efficacy in introductory programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. ICER '20: International Computing Education Research Conference. ACM, Virtual Event New Zealand, (Aug. 10, 2020), 158–169. isbn: 978-1-4503-7092-9. doi:10.1145/3372782.3406281.
- [73] John Sweller. 2011. Cognitive load theory. In *Psychology of learning and motivation*. Vol. 55. Elsevier, 37–76. Retrieved Dec. 28, 2023 from <https://www.sciencedirect.com/science/article/pii/B9780123876911000028>.

- [74] Karthikeyan Umapathy and Albert D. Ritzhaupt. 2017. A meta-analysis of pair-programming in computer programming courses: implications for educational practice. *ACM Transactions on Computing Education (TOCE)*, 17, 4, 1–13.
- [75] Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: integrating LLMs into CS1 instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITICSE 2024)*. Association for Computing Machinery, New York, NY, USA, (July 3, 2024), 297–303. isbn: 979-8-4007-0600-4. doi:10.1145/3649217.3653584.
- [76] Marcel Valový and Alena Buchalcevova. 2023. The psychological effects of AI-assisted programming on students and professionals. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME). ISSN: 2576-3148. (Oct. 2023), 385–390. doi:10.1109/ICSME58846.2023.00050.
- [77] Lev S. Vygotsky. 1978. *Mind in Society: The Development of Higher Psychological Processes*. Michael Cole, Vera John-Steiner, Sylvia Scribner, and Ellen Souberman, (Eds.) Harvard University Press, Cambridge, MA. isbn: 978-0-674-57629-2.
- [78] Matthew D. Webb. 2023. Reworking wild bootstrap-based inference for clustered errors. *Canadian Journal of Economics/Revue canadienne d'économie*, 56, 3, (Aug. 2023), 839–858. doi:10.1111/caje.12661.
- [79] Justin D. Weisz et al. 2021. Perfection not required? human-AI partnerships in code translation. In MAG ID: 3154248444. (Apr. 14, 2021), 402–412. doi:10.1145/3397481.3450656.
- [80] Yuankai Xue, Hanlin Chen, Gina R. Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT help with introductory programming?an experiment of students using ChatGPT in CS1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '24)*. Association for Computing Machinery, New York, NY, USA, (May 24, 2024), 331–341. isbn: 979-8-4007-0498-7. doi:10.1145/3639474.3640076.
- [81] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, 62–71.
- [82] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, 4, (Jan. 1, 2023), 100147. doi:10.1016/j.caeai.2023.100147.

- [83] Achim Zeileis. 2004. Econometric computing with HC and HAC covariance matrix estimators. *Journal of statistical software*, 11, 1–17. Retrieved Feb. 19, 2026 from <https://www.jstatsoft.org/article/view/v011i10/0>.
- [84] Achim Zeileis and Torsten Hothorn. 2002. Diagnostic checking in regression relationships. *R News*, 2, 3, 7–10. <https://CRAN.R-project.org/doc/Rnews/>.
- [85] Achim Zeileis, Susanne Köll, and Nathaniel Graham. 2020. Various versatile variances: an object-oriented implementation of clustered covariances in r. *Journal of Statistical Software*, 95, 1–36. Retrieved Feb. 19, 2026 from <https://www.jstatsoft.org/article/view/v095i01/0>.

Chapter 8

Conclusion

Chapter Contents

- 8.1 Dissertation Summary 318
- 8.2 Contributions 321
 - 8.2.1 Chapter 2: Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation 321
 - Overview 321
 - Findings 321
 - Intellectual Merit 322
 - Broader Impacts 322
 - 8.2.2 Chapter 3: Novice Interactions with GitHub Copilot Under Time Pressure: A Deductive Qualitative Interview Study 323
 - Overview 323
 - Findings 323
 - Intellectual Merit 323
 - Broader Impacts 324
 - 8.2.3 Chapter 4: Relationships Between Trust, Compliance, and Performance for Novice Programmers Using AI Code Generation 324

	Overview	324
	Findings	324
	Intellectual Merit	325
	Broader Impacts	325
8.2.4	Chapter 5: HBCU Student Perspectives on Identity, Persistence, and Code- Generating AI in CS Education: A Case Study	326
	Overview	326
	Findings	326
	Intellectual Merit	327
	Broader Impacts	327
8.2.5	Chapter 6: Beyond Efficiency: A Critical Political Economy Perspective on AI Code Generators in Computing Education	328
	Overview	328
	Findings	328
	Intellectual Merit	328
	Broader Impacts	329
8.2.6	Chapter 7: Fast and Forgettable: A Controlled Study of Novices' Perfor- mance, Learning, Workload, and Emotion in AI-Assisted and Human Pair Programming Paradigms	329
	Overview	329
	Findings	329
	Intellectual Merit	330
	Broader Impacts	330
8.2.7	Summary	331
8.3	Educational Recommendations	331
8.3.1	Approaches to Avoid	331
	Avoid Ban Policies and Forced Guardrails	332
	Avoid Free-for-all Policies and Ambiguity	334

	Avoid Banal and Anachronistic Policy Justifications	336
8.3.2	Approaches to Embrace	337
	Embrace Self-Imposed Guardrails and Accessible Alternatives	337
	Embrace Layered Coursework (Charge-Challenge-Check)	339
	Embrace Policy Co-Creation and Timeless Justifications	341
8.4	Limitations	342
8.4.1	Threats to Internal Validity	343
	Qualitative Trustworthiness	343
	Statistical Validity	344
	Endogeneity	346
8.4.2	Threats to External Validity	347
	Ecological Validity	347
	Sampling	349
8.4.3	Threats to Construct Validity	350
8.5	Future Work	350
8.6	Closing Thoughts	352
	References	353

8.1 Dissertation Summary

Code-generating AI has disrupted computing education by providing never-before-seen abilities to automate problem-solving and computer programming. Educators have taught students with consistent and refined approaches that stood the test of time through numerous fads and economic circumstances. However, the simultaneous hope, fear, excitement, and grief associated with AIDEs like GitHub Copilot and ChatGPT have left teachers and leaders in education searching for answers about how best to integrate and regulate AI in pedagogy. This dissertation provides much-needed empirical foundations for CS education policy by exploring the AI-assisted programming paradigm and comparing it to traditional programming approaches. To explore the

overall research question laid out at the outset of this dissertation, restated below, this work was organized under four increasingly focused aims. First, quantitative and qualitative approaches evaluated the use of GitHub Copilot for Python programming against a solo programming control, examined the interaction dynamics and human impacts of GitHub Copilot's presence during time-pressured programming, and modeled a theoretical cycle of trust, compliance, and performance derived from the human factors literature. Second, qualitative explorations critiqued the societal impacts of AIDEs in relation to participants' actual programming experiences, computing education success determinants, histories of identity-based social exclusion, and U.S. economic politics. Third, a mixed-methods investigation tested the substitution of a human teammate with Copilot in terms of not just performance, workload, and emotion, but also learning retention. Fourth and finally, the present chapter provides six concrete Educational Recommendations, justified by assertive arguments about the findings of the preceding aims. These guidelines, as well as the various contributions revisited in this chapter, have satisfied the original goal of this work to provide evidence-informed guidelines as a foundation for next-generation AI policies in computer science education. As a reminder, the following research question broadly describes the scope of this work.

Overall Research Question What does studying novice programmers in solo, AI-assisted, and human pair programming paradigms reveal about how computing pedagogy can best promote student success in the age of AI?

The points below briefly review the investigations that carried out each of the four overarching aims. Note how the scope of work narrowed from wide areas of inquiry to narrow questions with simplified purpose. As the work progressed, the findings of early studies began to clarify that GitHub Copilot was clearly useful and helpful to novices, but concerns began to mount regarding its affective influence. By the time of planning Chapter 7, it was clear that a focus on emotion and learning would be critical to incorporating the overall findings into informed recommendations. Each chapter is noted below, but the empirical investigations are also more thoroughly rehashed in the section that follows.

- **Aim 1: Evaluate & Explain Human-AIDE Interaction**
 - Chapter 2 (Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation) evaluated GitHub Copilot by comparing it directly to solo programming using a within-subjects quantitative experiment.
 - Chapter 3 (Novice Interactions with GitHub Copilot Under Time Pressure: A Deductive Qualitative Interview Study) explained how novices use AIDEs under time pressure and how doing so impacts their thoughts and feelings.
 - Chapter 4 (Relationships Between Trust, Compliance, and Performance for Novice Programmers Using AI Code Generation) provided a quantitative explanation of AIDE trust dynamics, bridging between the human-automation interaction literature and AIDE interaction research.
- **Aim 2: Situate Human-AIDE Interaction in a Societal Context**
 - Chapter 5 (HBCU Student Perspectives on Identity, Persistence, and Code-Generating AI in CS Education: A Case Study) examined issues of identity and persistence in relation to code-generating AI through by gathering the perspectives of HBCU students after they programmed with Copilot.
 - Chapter 6 (Beyond Efficiency: A Critical Political Economy Perspective on AI Code Generators in Computing Education) used Critical Political Economy theories to critique capitalist values that create hazards for computing students in their interactions with AIDEs.
- **Aim 3: Compare Human-AIDE Interaction to a Social Alternative**
 - Chapter 7 (Fast and Forgettable: A Controlled Study of Novices' Performance, Learning, Workload, and Emotion in AI-Assisted and Human Pair Programming Paradigms) evaluated GitHub Copilot as a substitute for a human teammate, with particular attention to affective and learning effects.

- **Aim 4:** Synthesize Findings Into Computing Education Recommendations
 - Chapter 8 (this Conclusion chapter) brings the preceding chapters 2 to 7 together into actionable takeaways for educators. After a recap of the merit, findings, and impact of each of the empirical investigations, specific Educational Recommendations for institutional policymakers and classroom teachers are synthesized in Section 8.3.

8.2 Contributions

Though the Educational Recommendations below are a concrete outcome of this dissertation, the six empirical investigations up to this point contributed key foundational research about the human implications of code-generating AI. This section focuses on the intellectual merit and broader impacts of Chapters 2 to 7.

8.2.1 Chapter 2: Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation

Overview

This study compared objective performance and subjective measures of workload, emotion, and self-perceived performance between solo and AI-assisted programming conditions to evaluate Copilot for use by CS students.

Findings

GitHub Copilot significantly improved programming performance and reduced mental workload and effort, but observed differences were not significant for other dimensions of workload (physical, temporal, performance, frustration), emotion (valence and arousal effects), or self-perceived performance. Between replicates in each condition, however, both actual and self-perceived performance improved significantly for AI trials but not for solo trials.

Intellectual Merit

Early research found that AI was effective for a range of programming tasks [18, 78, 111], but when studies began including human users, the benefits were less clear [54, 110, 104]. Not only did this study cement the performance benefits of AIDEs for novices, it shed light on the uncertain well-being [47] impacts of AI code generators. This human-centric focus has played an integral part in a budding knowledge base on the affective harms of AIDEs compared to traditional paradigms [112, 27, 71, 44].

Broader Impacts

Providing quantitative evidence about the two-sided nature of programmer-AI interaction is essential to prevent unfounded policies that are harmful to students. Showing the benefits of AIDEs to performance and workload shows that ban policies are unrealistic for the long term, while reporting inconclusive findings about emotion and self-efficacy can slow over-exuberance to adopt AIDEs in CS education. Learning to program has a reputation for being very challenging [8, 50, 67, 90], so we should take any opportunity to reduce barriers to success. This is especially important for female and racially underrepresented students, who can face amplified barriers compared to non-marginalized students [4, 6, 23]. On the other hand, traditional social supports are known to be powerful tools for student success [6, 19, 4, 9, 88], and the lackluster pseudo-social diversion of generative AI threatens to distract students from these supports. Balancing the productivity benefits of automation with null or negative effects on well-being is also instructive to other types of labor such as journalism, medical decision-making, and agriculture.

8.2.2 Chapter 3: Novice Interactions with GitHub Copilot Under Time Pressure: A Deductive Qualitative Interview Study

Overview

This study merged a broad review of the qualitative human-AIDE interaction literature with novel interview data from novice programmers after they programmed in solo and AIDE-assisted paradigms under time pressure. A set of themes was developed by deduction from existing work and induction from novel findings to describe human-AIDE interaction, explain it, and reveal its impacts on the programmer as a human.

Findings

We found that programmers had strong values about how to use AIDEs responsibly and worked strategically and observantly with Copilot, resulting in frequently harmonious interactions. However, participants also fell into aimless, submissive interaction strategies that often caused ineffective collaboration, dispassion, or negative feelings.

Intellectual Merit

Many prior studies examined programmers using code-generating AI from a Human-Computer Interaction (HCI) perspective [55, 62, 76, 104, 3, 85, 49, 51, 93, 108, 31, 10]. However, these studies often sampled intermediate to advanced programmers and allowed programmers to use AIDEs with no time pressure or basis for comparison. Furthermore, many of these studies arrived at similar findings, but there was a need to cross-reference them for overlapping takeaways. Our study replicated many findings of prior studies, but also diverged to show more maladaptive tendencies. We argue that the focus on novices and the imposition of time pressure bridged beyond the initial novelty of the technology and helped participants articulate some deeper harms of AIDEs in their interviews.

Broader Impacts

Quantitative research often lacks the nuance of qualitative work with respect to individual differences and potent but uncommon experiences. Negative-self talk (i.e., shame and low self-efficacy) were features of many participants' experiences that were not captured by the conclusions of Chapter 2. Representing these harms, in addition to the loss of metacognitive control we observed, help to argue against free-for-all AI policies in computing education. It is critical that in adopting automation intended to benefit students, we do not inadvertently expose them to undue emotional harms [47]. It is easy to think that these harms are confined to students or developers, but software production touches nearly every industry. Harming programmers, therefore, threatens to upset a range of production pipelines for numerous essential and life-saving technologies.

8.2.3 Chapter 4: Relationships Between Trust, Compliance, and Performance for Novice Programmers Using AI Code Generation

Overview

This study looked at the associations between trust in GitHub Copilot, how much novices used its compliance features, and programming performance. There was an expected cycle, starting at trust, leading to rate of compliance, impacting performance, and feeding back to adjustments in trust. Quantitative analysis of these measures over time revealed the extent to which this cycle held up within the experimental context of Chapters 2 and 3.

Findings

A factor analysis demonstrated the relevance and utility of the Muir [75] trust scale for human-AIDE interaction research. Regression analysis was inconclusive regarding the impact of trust on the number of compliance events, likely due to methodological constraints and a small true effect (if present). However, greater compliance was associated with stronger performance, and stronger performance led to higher subsequent levels of trust.

Intellectual Merit

Substantial work had theorized [91, 40, 61] about cycles of trust, compliance, and performance in the human-automation interaction context. Empirical work has also generally supported a positive feedback cycle from trust to compliance to performance and back to trust [79, 21, 74, 109, 73]. Quantitative trust research in the specific context of human-AIDE interaction, by contrast, is still in its infancy [2, 83, 98]. Though research in the field suggests that a trust-compliance-performance cycle could reasonably carry over to programming with AI [49, 83, 93, 104, 83, 30, 29], we were not aware of any prior quantitative efforts to check. This study provided a foundational look at the role of trust in human-AIDE interactions that warranted further study into the topic for future researchers.

Broader Impacts

If trust determines how much programmers use AI and training interventions can influence trust, then educators can design better training materials for optimizing the performance and safety of human-AI programming teams. As noted earlier, the ability to produce high quality software impacts a myriad of real-world contexts, so understanding how to optimize programmers' use of AI is broadly important. Beyond this, though, true performance is easily misrepresented by common metrics relating to the number of production milestones reached in a given period time. For example, bugs and security vulnerabilities in code can be missed by such metrics, but can degrade the societal impact of software. Take the case of CVE-2004-0113, a trigger for memory leaks in the extremely popular Apache HTTP web server [77]. The consequence of this bug was that remote attackers could intentionally balloon the memory usage of the application, causing performance degradation and denial of service for legitimate users. As a result, high stakes web services making use of Apache such as emergency response or crime reporting systems, could have been threatened by bad actors. Being trained on imperfect human code, GitHub Copilot and other AIDEs can introduce security vulnerabilities like CVE-2004-0113 that may go unnoticed by human programmers [80]. Therefore, exploring whether trust-based or other training

interventions can impact the extent to which developers comply with and rely on AI suggestions is critical for preventing seemingly reliable software systems from failing when edge cases and attack vectors come to light.

8.2.4 Chapter 5: HBCU Student Perspectives on Identity, Persistence, and Code-Generating AI in CS Education: A Case Study

Overview

This study collected the perspectives of students at a Historically Black College/University to examine personal and computing identity, helpful and harmful contributors to students' success in computing education, and perceptions about how AIDEs might fit into this landscape. Participants shared their perspectives immediately following solo and AI-assisted laboratory programming, so their thoughts were grounded in concrete experiences rather than merely speculation.

Findings

Allowed to define their identities however they chose, participants focused mostly on their humanity (e.g., curiosity, creativity, and craftsmanship) as well as their desires to have an impact on their peers and the world at large. They found that positive feedback, social support systems, extracurricular learning, and financial aid were helpful to succeeding in CS, while difficult coursework, deficits relative to others, and social barriers were harmful. We determined that AI is helpful for promoting the success of a range of computing students by providing moral support, reducing perceived judgment, and compensating for deficits; it is also harmful because it shows students their weaknesses, disrupts their personal identities, interrupts the learning process, and discourages positive social interactions.

Intellectual Merit

The research community has recognized the value of studying the equity implications of AI code-generators, but little work has taken concrete steps to do so [60, 86]. This study made an intentional effort to include the voices of Black students, who are underrepresented in U.S. computing education compared to students of other races [22, 16, 17]. The work also corroborated existing understandings about how to support computing students' academic success overall [100, 41, 25, 4, 9, 42, 6, 19, 89, 23]. With respect to AI, we identified concrete intersections with identity and persistence factors that provide foundational knowledge for educational policymakers.

Broader Impacts

The impact of this work was giving Black HBCU students a voice in the conversation about code-generating AI in computing education [88, 87]. Frankly, the political climate in the United States at the time this chapter was written was openly hostile to inclusivity-focused research. However, this study demonstrates how pedagogical research that aims to include underrepresented groups leads to universally helpful insights [99]. Most findings from this study were not specific to Black students, though all participants were Black. Just as designing products to be more useful to users with varied physical capabilities begets benefits for all users, designing pedagogy to be inclusive of minoritized students can lead to improvements in educational quality for all students. Readers should not consider Black students to be victims in need of special help from AI compared to non-Black students. Instead, Black students are simply regular participants in computing education who deserve to be included in the conversation about how best to be supported in their educational journeys. It should not be controversial to ensure that a diversity of students inform wide-reaching policies.

8.2.5 Chapter 6: Beyond Efficiency: A Critical Political Economy Perspective on AI Code Generators in Computing Education

Overview

This study used sociological theories of Critical Political Economy and interview data from the studies described in Chapters 2 to 5 to explore and critique the underlying capitalist values borne out in computer science students' experiences programming with GitHub Copilot under time pressure.

Findings

We found that participants' values reflected rationalization—the idea that calculable and objective performance measures are better than subjective experiences—and automation bias—an assumption that using Copilot would be more efficient than trying problems themselves. Participants also represented values of individualism, taking personal liability for Copilot's performance, and experienced alienation, being pulled away from enjoyment and social interest by Copilot's presence.

Intellectual Merit

Social theory is underutilized in computing education research despite many relevant explorations of its pedagogical implications [46, 53, 106, 52, 26, 88, 96, 100, 57, 59, 70]. Sociological scholarship, by contrast, has explored various aspects at education's intersection with labor economics and politics [37, 45, 15, 32, 11, 94, 38, 94, 13, 1, 28, 37]. This study bridges this gap by describing the entanglement of the economic incentives of software production with computing education and code-generating AI. Rather than relying only on theory or speculation, the concrete experiences of novice programmers using GitHub Copilot reveal novel social hazards associated with introducing AIDEs to CS education.

Broader Impacts

This study challenges common assumptions about the utility of AI in software development and computing education, offering educators alternative perspectives by which to guide their students. The intended political stance is neutral, but naturally resists capitalist thought as a polemic exercise. Offering future U.S. laborers alternative narratives about their humanity and purpose helps empower them to stand up for their rights to happiness and welfare [47]. Education has wide-reaching potential to reduce wealth inequality because it recycles desires for working-class citizens to shape the use of AI technology in their own best interests [13]. Automation has displaced and disturbed laborers from other industries in the past [101] and will continue to do so for unknown industries in the future. The current response of CS educators to AI code generators may well shape the future trajectories of other markets yet to be disrupted.

8.2.6 Chapter 7: Fast and Forgettable: A Controlled Study of Novices' Performance, Learning, Workload, and Emotion in AI-Assisted and Human Pair Programming Paradigms

Overview

Given the accumulation of evidence from the preceding studies that Copilot's help is at least impersonal if not actively anti-social, this study compared it directly to a social human partner. Furthermore, a novel design incentivized participants to balance their programming performance with understanding of their code, which was quantified using an individual retest following a one week retention interval.

Findings

We found that participants performed significantly better with Copilot compared to their human teammates, and their mental workload, temporal workload, and effort was significantly lower, though frustration differences were inconclusive. Participants' learning retention was not sig-

nificantly different between conditions, but performance did trend worse on tasks learned with Copilot compared to a human partner. This was especially true for those who were the stronger partner within their human team. Finally, the emotional effects of the programming experiences were significantly more positive and arousing in the human-teaming condition compared to the AI-assisted condition.

Intellectual Merit

Pair programming is recognized as a positive social approach to supporting students through the challenges of learning to code [97, 39, 72, 103, 35, 34]. The introduction of GitHub Copilot and other AIDEs led to a popular notion that AI assistance was comparable or substitutable for human collaboration [33, 48, 20, 69], supposedly carrying many of the same benefits without the same social and logistical drawbacks [97, 39, 97]. However, when this work was begun, little had experimentally compared the two paradigms [58, 48, 69]. Further investigations have since been published [105, 68, 82, 27], but they have not shown overall dominance by either paradigm. For example, [82] found participants preferred human tutors for high-stakes questions, and [27] found that a human-human teaming condition and human-AI teaming condition comparably improved performance over a solo condition, but participants experienced lower social presence with AI. Most importantly, while some of these studies have seen students gain performance benefits on academic assignments from AI, none to our knowledge has compared unassisted retest performance [105, 68, 82, 27].

Broader Impacts

The impact of this work is advocacy for preserving human-to-human connection even as we adopt AI into everyday activities. The results of this study showed clear practical benefits to AI alongside clear emotional benefits to pair programming. Further, the inconclusive effect on learning points to the fact that neither assistance paradigm ought to be used alone. We should introduce code-generating AI in CS education without trying to replace human interactions that

have been supporting our well-being for thousands of years. Showing that there are distinct emotional advantages to human-human programming is a reminder to all intellectual laborers that some aspects of their work should probably never be replaced—even if they can be.

8.2.7 Summary

Chapters 2 to 7 described six empirical investigations contributing both quantitative and qualitative findings that contributed to knowledge bases in computer science, human factors, sociology, and education. Overall, we found that GitHub Copilot is best used by computing students in moderation and in conjunction with social interventions. AIDEs give clear performance and workload advantages over solo and paired paradigms but create undeniable harm to students' autonomy and self-perceptions and lack the affective properties of social interactants. The impact of this work extends beyond education policy into software safety, labor rights, educational equity, and general philosophy.

8.3 Educational Recommendations

The culmination of this work is Aim 4, which set out to integrate the findings of Aims 1-3 (summarized in Section 8.2 above) into straightforward and actionable recommendations for computer science educators. This section sets out three concrete approaches to avoid and three concrete approaches to embrace. Table 8.1 summarizes these recommendations. The subsections that follow provide examples of each proscription and each suggestion well as evidence-informed arguments supporting these recommendations.

8.3.1 Approaches to Avoid

Extrapolating from the findings of this dissertation and those of related research in the field, this section provides cautions against policy approaches that are likely to work against fairness and the best interest of students. Such approaches include overly conservative or liberal policies,

Table 8.1: Overview of computing education recommendations developed for Aim 4.

Approaches to Embrace	Approaches to Avoid
Self-Imposed Guardrails and Accessible Alternatives	Ban Policies and Forced Guardrails
Layered Coursework (Charge-Challenge-Check)	Free-for-all Policies and Ambiguity
Co-Creation and Timeless Justifications	Banal and Anachronistic Justifications

especially when accompanied by weak justifications provided by educators to students in a one-way, top-down manner.

Avoid Ban Policies and Forced Guardrails

Examples Ban policies are restrictions at the classroom or institutional level which prohibit students from either using certain features of GenAI, such as code generation, or disallow GenAI use entirely. This also includes well-intentioned policies that pass judgment on certain modes of AI use such as: "It is okay to use AI tools to explain concepts, but you may not use them to replace critical thinking." Tools with guardrails (e.g. [64, 56, 66]) fall into this category unless students are expressly allowed to use commodity tools as well, making their guardrails optional.

Argument Prohibition policies slow society's overall pace of adaptation to AI innovation and are unlikely to be followed. They also create moral hazards for students, cause inconsistencies between settings with and without such policies, and miss opportunities to promote student success and educational equity. Tools with guardrails offer students agency to choose adaptive patterns over maladaptive ones, but mandating overly restrictive guardrails will simply drive students to covertly use general-purpose AIDEs.

Evidence AI offers clear benefits to programming performance that neither students nor their future employers will be able to ignore [54, 68, 27]; banning AI entirely is both unrealistic and out of touch with the reality in which computing students live [113, 86]. In Chapter 2, we showed that GitHub Copilot significantly improved novices' programming performance compared to solo pro-

programming, and in Chapter 7 we showed Copilot was also more helpful to participants' programming performance than human teammates. Chapters 2 and 7 also showed significant differences from control in several non-emotional workload dimensions, indicating that Copilot reduced the difficulty of programming compared to both solo and human-human pair programming paradigms. In Chapter 4, we found a significant positive relationship between compliance to Copilot's suggestions and programming performance, meaning Copilot can offer increasing benefits as users accept its help more readily. We also know from Chapters 3 and 6 that students have core intentions to use AI responsibly, but need support to stay out of tempting over-reliance traps. As discovered in Chapters 2 and 3, using AIDEs effectively is an acquired skill and not one students can be assumed to have naturally. Based on the economic imperatives focused on in Chapter 6, students will most likely be expected to use AIDEs in their careers. Therefore, it is critical to teach students how to harness the clear utility of AI code generation technology, rather than assuming that other benefits of traditional paradigms can outweigh the massive performance benefit of using AI to one's advantage [27, 86]. One major disadvantage to mandating guardrails against code generation is that it leaves students unprepared to use code-generators in their careers. Finally, the findings of Chapters 3 and 5 emphasize that AIDEs act as learning resources and moral supports to students as they progress through the challenging aspects of computing education. Prohibiting their use stands to harm struggling students most, as strong students are poised to succeed regardless of the rules they must navigate [71, 84, 54]. Doing so also worsens educational inequity because, though students at a given institution or in a given course may all be subject to the same restrictions, those in other courses or at other institutions with more liberal policies will gain a competitive edge in the labor market [60]. Furthermore, as noted in Chapter 5, using AI assistance as an educational accommodation is an opportunity to level the playing field for students with unique disadvantages like mental illnesses or lack of access to education and technology during primary and secondary education.

Avoid Free-for-all Policies and Ambiguity

Examples Free-for-all policies are implied by a lack of relevant policy and also include explicit policies allowing students to use code-generating AI for any purpose. They also include policies such as: "You may use AI to support you in any way, but you must submit an explanation and reflection on how you used AI for any given assignment that includes transcripts of your conversations with AI." Ambiguous policies are those that, intentionally or unintentionally, are vague enough to be open to interpretation. For example, the policy, "Only use AI in ways that will support your learning," will leave students questioning what they are and are not allowed to do.

Argument Free-for-all policies expose students to risks associated with heavy AI usage and ambiguity creates confusion. Students may lack awareness of their true competence if they are always using AI assistance, and their capability to perform may be diminished if AIDEs underperform or become unavailable. Ambiguous policies and additional assignment components seem like they hedge against unforeseen AI harms, but they ultimately sow confusion and discontent, disproportionately punishing the most honest students.

Evidence Our findings suggest that AI's strong performance benefit is ephemeral, fading when the assistance is removed, but its harms have broader reach. In both Chapters 2 and 7, programmers performed worse alone and in teams with other humans than they did with Copilot. In Chapter 7, when we attempted to compare the learning effect of Copilot to that of a human teammate, the result was inconclusive but trended toward being worse with AI. Though our work did not attempt to compare students' perceptions of competence to their actual competence, prior work has suggested that AI deludes students into thinking they are more knowledgeable than they really are [84]. While it is unclear whether AI is better or worse for learning than traditional paradigms, other harms we identified point to overly permissive policies being hazardous. Chapters 2, 3 and 5 to 7 showed that Copilot brought a fresh perspective and vast knowledge base that would clearly contribute to learning; however, it also lacked emotional resonance, robbed programmers of their autonomy, interrupted their personal identities, drew them away from social help-seeking,

and encouraged negative self-talk. These harms are enough to dissuade free-for-all policies in favor of hybrid policies that permit the use of code-generating AI in some circumstances but not others.

Another consideration is AI access. Participants in Chapter 5 reported unwillingness to pay for AIDEs as long as relatively comparable free alternatives exist. These students, studying at a state university, with some having been raised in poor families lacking technology access, may have different thresholds for limiting costs as compared to other students. As an example, the pricing tiers for ChatGPT¹ currently range from \$0 to \$200 per month. For a wealthy student hoping to gain an advantage, the top tier price may be negligible. Just as standardized test preparation courses, fencing lessons, private tutoring, or Chegg² subscriptions might provide wealthy families opportunities to give their children educational advantages, premium AI can do the same. If an institution or instructor chooses not to address code-generating AI or take a stand on how they prefer their students use it, they risk a massive gap forming between students with high AI access and those with low AI access [102]. By contrast, encouraging students to use tools freely available to all of them gives a more even distribution of the potential advantages among students from different backgrounds.

Our findings also suggest that pairing free-for-all policies with AI usage reflections is insufficient as a policy response. Chapter 5 delves into what makes it hard to persist in computer science education at the post-secondary level. Two major challenges are the difficulty (or at least perceived difficulty) of coursework and lack of applicability (or perceived applicability) to real-world problems. Due to the tendencies observed in Chapters 3, 5 and 6 for programmers to fall into mindless and submissive roles with GitHub Copilot, having students reflect on their AI usage is likely to help them regain metacognitive control. However, pairing this technique with a free-for-all policy creates similar moral hazards to a ban policy. A dishonest student seeking to earn high grades as easily as possible becomes incentivized to complete work with AI assistance but present the work as unassisted to avoid the extra effort required to produce an AI reflection.

¹<https://chatgpt.com/>

²<https://www.chegg.com/>

Once again, honest students will be penalized with what is arguably "busy work," increasing the difficulty of completing coursework without having much applicability to a real-world activity. Because of the economic incentives described in Chapter 6, employers are not likely to allow time for such reflection opportunities, let alone require them. A potential solution is to pair reflection activities with AI-required assignments, where all students must do the activity because all were required to use AI for it. In this case, honest students do not face the same disadvantages. In summary, reflections have strong potential but pair poorly with free-for-all policies.

Avoid Banal and Anachronistic Policy Justifications

Examples Some educators will feel pressure to educate their students about AI, but will not have the time, knowledge, or willingness to engage in nuanced discussions on the topic. A banal or trite warning to students might go like, "AI can be helpful, but you always have to check it because it makes mistakes." An anachronistic (i.e., outdated or temporally-restricted) caution would focus on limitations of the current time (or a past time), failing to acknowledge how innovation has or could challenge the validity of the caution in the future. For example, saying "ChatGPT produces more memory leaks in C code than human programmers," could be true at a given time but cease to be true in the future. Any argument discouraging use of ChatGPT on this account, would then be invalidated by such a change.

Argument Lazy cautions against code-generating AI are worse than no cautions at all. Students are capable of having refined discussions on the harms and benefits of AI. Policy justifications should rest on timeless truths or up-to-date knowledge rather than hackneyed warnings lacking in depth or basis in fact.

Evidence Qualitative analyses in Chapters 3, 5 and 6 showed that students hold strong moral convictions about how to responsibly use AI code generators, but their behavior can diverge from these convictions. The sociological critique in Chapter 6 details the political-economic roots of participants' positioning of AI within their concepts of the labor process. Participants felt liable

for the failings of AI but would not take credit for its success (see also Chapters 2, 3 and 7). Sometimes, as noted in Chapters 3 and 5, participants misunderstood AI as being an expert, authority, or database of facts. Telling students that AI is imperfect is not enough to reconstruct these misperceptions. Rather, students deserve thoughtful education about the philosophical underpinnings of their involvement in the software development process. Chapter 1 argues in Section 1.2 why we ought to continue to educate computer scientists in the age of AI. Chapter 6 further challenges the artifacts in students' thought patterns of outdated or politically biased paradigms for responsibility in the world of computing. Overall, our findings suggest that the basic premise that computer scientists need to exist must be justified for students early in post-secondary programs. Simply warning of potential misdirection or problems in the media will leave students without appropriate focus on the creative, value-centric, and communal ideals they need for fruitful careers in the modern day.

8.3.2 Approaches to Embrace

The following suggested approaches are based on the body of work in this dissertation, which also draws on dozens of prior studies in the field (see Sections 1.4 and 8.2). Findings support a partial adoption approach where students are encouraged, not forced, to curtail their AI use, while still using it in clearly defined cases. Co-creating policies with instructors, who can facilitate discussion about the deeper purposes for regulation, allows students to dial up the restrictiveness of their own policies to meet their collective learning objectives. These recommendations are also informed by the teaching experiences of the author, who has found that students who have agency in the classroom rule-making process are more satisfied with those rules over time.

Embrace Self-Imposed Guardrails and Accessible Alternatives

Examples Self-imposed guardrails are decisions made by students about which AI tools and features to use as well as customizations to those AI tools that change how they work. For example, while the CS50 Duck Debugger has instructor-provided guardrails against code generation

[66], students given the option to use ChatGPT over the Duck can still make a conscious decision toward the safer option on their own. Customizations might include Copilot's snooze feature, which turns off inline code completion for 5 minutes or Copilot's instructions feature, which lets users tell Copilot about their background or provide general instructions for how it should behave in chat conversations. Within instructions files, students could describe their programming expertise, their language background, their learning goals, and ask Copilot not to include complete code examples in chat responses. This way, when working on AI-inclusive assignments, students can learn how to mold their AI's behavior to their current goals, rather than relinquishing control over the interaction trajectory.

Accessible alternatives are ways to attract AI-agnostic help-seeking such as internet searching, pair programming with peers, office hours with teaching assistants, and instructor-led review sessions. Instructors should be looking for ways to make non-AI based help-seeking so effortless and fruitful that students are least attracted to use AI in times of crisis. For example, many post-secondary programming courses include laboratory sections that instructors can use to build scheduled pair programming into students' weekly routines. Additionally, making course resources like lecture slides and example code easily accessible and clear to understand gives alternatives to AI as a default [43].

Argument Computing pedagogy must gently nudge students toward adaptive learning interactions with AI code generators. Dictating how students should use AI leaves them out of the conversation and disregards their own extant morality. If instructors want students to use AI in certain ways or avoid it altogether, they should make those behaviors extremely attractive and as effortless as possible.

Evidence One major finding of Chapter 3 was the tendency for novices to fall mindlessly into maladaptive behaviors with AI, in line with prior work [71, 84]. Deep down, participants desired to learn and had no malicious intent overall (see also Chapter 6); they simply felt drawn into concerning patterns by time pressure and a lack of metacognitive awareness. The takeaway is

that adaptive student behaviors need to be made so seamless as to be practically unavoidable. If resorting to ChatGPT seems easier or more preferable than an instructor's recommended approach, students will undoubtedly go that route [56]. Just as adding extra optional steps for AI users under laissez-faire policies encourages dishonesty (see Section 8.3.1), encouraging students to use AI with guardrails imposed by the instructor conflicts with student agency and buy-in. Students need to choose guardrails because they want them—not because they are required. Furthermore, there needs to be instructional time dedicated to teaching students how to control AI tools, rather than assuming they will invest that time on their own. GitHub Copilot has a broad and ever-changing set of features that most users never even try. As we found in Chapter 3, salient user-interface elements dominate human perception and guide behaviors toward the most obvious features. The findings of Chapters 2 and 3 showed that participants took time to learn how to collaborate with Copilot effectively. If students are expected to write system prompts for Copilot, turn off code completion in certain situations, and maintain agency in conversations with ChatGPT, they must be trained on how to do so [24]. It is also critical that instructor-provided AI tooling is always as good or better than off-the-shelf tooling [56]. There should be clear explanation provided for students about how and why a bespoke tool can be more useful for them than a commodity option. Otherwise, because of the pressures described in Chapter 5, they will simply drift back to a superior alternative in times of crisis. In summary, guardrails and AI alternatives have their place, but students must choose them naturally and not by force.

Embrace Layered Coursework (Charge-Challenge-Check)

Examples *Charge-Challenge-Check*, introduced here, is a suggested approach to the coursework in a computing course that employs three levels of AI usage for assignments (i.e., coursework layering). Instructors should strive to use all three layers to some degree within each course, based on students' collective learning objectives. At one end of the spectrum, *Charge* activities should give students free license to use AI for a maximum productivity effect; for example students might use AIDEs to plan, produce, and test a complex application in a short time frame. In the middle, *Challenge* activities glorify slowness and reward students for displaying authority

over AI tools, critiquing assumptions and details, and revisiting sloppily produced codebases. Examples include required engagements with AI to interrogate algorithms or identify security vulnerabilities in code with known issues. Both *Charge* and *Challenge* require students to be given access to best-in-class tools, including mid-tier premium options for which institutions can purchase subscriptions at scale. This is no different from library subscriptions to academic journals or licenses purchased to give students experience with 3D modeling software. At the far end, *Check* assignments go back to basics, testing students on the skills and knowledge they retain for unassisted scenarios. This could be anything from paper tests to proctored individual programming assignments to interviews with instructors.

Argument Far extremes are usually worse than hybrid approaches. *Charge-Challenge-Check* rests on the assumption humans should have some ability to solve problems and write code without AI assistance, but that using AI to its fullest advantage is also a professional duty. The intermediate space is driven by philosophical ideals that challenging extremes and engaging in deliberate slowness are beneficial to society and the human experience.

Evidence As mentioned in Section 8.3.1, there are legitimate reasons to avoid both outright bans of AI technologies and complete policy agnosticism. Participants' subjective reports in Chapters 3, 5 and 6 show that students are wary of the harms and temptations of AIDEs, while simultaneously realizing their historical significance and utility in software development. Because of the economic and political systems discussed in Chapter 6, each new generation of computing students has the power to perpetuate or change norms about how AI should be used in the software industry. However, the clear performance benefits of AI code generators reported in Chapters 2, 4 and 7 are too strong to be ignored, so sheltering students while they are in school will not likely have its intended effect. Refusing *Charge* assignments not only leaves students unprepared for skillsets their employers want them to have, it also means they will make their first mistakes using AI as a labor accelerant in the much higher-stakes environments of the real world. Leaving out the *Check* component may lead to an overconfident workforce unequipped for

the inevitable complications associated with imperfect automation reliability. Chapter 7 showed that solo programming assignments can ground programmers back to reality regarding what they really know versus what they think they know. Finally, Chapters 3 and 5 solidified the benefits of AIDEs for teaching, exploring new ideas, and providing moral support in CS education. In Chapter 6, we discussed how centering student autonomy and challenging assumptions about the inherent value of speed and productivity can complement these benefits and avoid harmful metacognitive and psychological disruptions. Overall, layering pedagogy into distinct buckets with clear rules that cover a comprehensive range of approaches is an antidote to the confusion and moral hazards warned about in Section 8.3.1.

Embrace Policy Co-Creation and Timeless Justifications

Examples Co-creation is the process of including students in the policy design process instead of deciding what is best on their behalf. As part of this endeavor, educators can guide discussions with students toward timeless moral and philosophical arguments they may not have come up with on their own. Timeless justifications are those that hold up regardless of the technical limitations of the day. These are necessary when enforcing any level of restriction from the educator side of co-creation to secure student buy-in. Timeless justifications for *Charge-Challenge-Check* coursework include the importance of developers' happiness and enjoyment in their work, communal responsibility for software safety, and considering environmental and quality-related externalities of hastily produced code.

Argument Policy co-creation based in non-technical, open discussions with students is the remedy for out-of-touch, tired warnings that justify AI resistance with shaky logic. Students will be better educated by inclusion in efforts toward responsible use of AIDEs. Students will still create sufficiently restrictive policies to protect themselves but will increase the practical relevance of education policy to everyday learning activities.

Evidence Chapters 3 and 6 revealed that novice programmers feel substantial moral obligations to use AI carefully and protect their ability to learn. However, as argued in Chapters 5 and 6, AIDEs trigger students to feel apathy and discontent with their involvement in the software development process. Realizing that AI can effortlessly replace many of the skills you so proudly develop as a computer science student is disheartening. There needs to be an open dialogue between students and educators about the new fears AI-based programming automation introduces. Those fears must be validated but also challenged through deeper discussions about what justifies continued human involvement in this industry. Co-creation draws on the visceral vulnerability of students facing this new career landscape. Drawing from research in the realm of public health, the acceptability of different COVID-19 response interventions varied widely across demographics of age and health status [12]. Therefore, if older adults created policies they found acceptable, they risked young people disagreeing with the policies and thus failing to adhere. Similarly, if a generation of educators creates policies reflecting the norms and pressures of their time, they risk writing regulations that misalign with students' active needs. Stepping back, the findings in Chapter 7 stress how AI resistance can promote developer well-being. The participants there were happier to work harder and perform worse with human teammates over Copilot. To assume that students will not realize their own deep-seated preferences for human connection sells them unfortunately short [52]. The findings of [27] further support this claim; they found the human-human paradigm led to a greater sense of collaboration and social presence compared to the human-AI or solo human paradigms. Capitalistic obsession with productivity distracts from the inherent need for laborers to feel excited to exist and work [57, 59, 70]. If AIDEs cause us more suffering than they avoid, their adoption is frivolous. Allowing students to take this stand and advocate for their own preservation is thus a strong approach for AI regulation [52].

8.4 Limitations

This section discusses some of the key limitations of this work with respect to internal, external, and construct validity. Additional discussion of limitations and threats to validity is provided in the

preceding chapters, but the overarching caveats are investigated here.

8.4.1 Threats to Internal Validity

The internal validity of this work is the extent to which the reported evidence accurately represents causal conclusions made about our samples of 17, 10, and 22 participants under the conditions created in the laboratory.

Qualitative Trustworthiness

In the case of chapters with qualitative components (Chapters 3 and 5 to 7), internal validity is threatened by our human limitations and biases as researchers. Rather than attempting to neutralize our biases, we acknowledged their presence and embraced the breadth of our qualitative data rather than trying to over-systematize it [14, 5, 65]. The narrow scope of what one group of researchers can read, know, or consider is biologically constrained. To mitigate threats to the validity of our findings, we took several steps. First, having multiple researchers participate in the analysis processes helped to challenge singular perspectives of this dissertation's author. Across chapters, more than ten individuals participated in the interpretation of qualitative datasets. Second, heavy reliance on prior qualitative work situated the findings against broader perspectives from the research community. Third, direct quoting was used in all qualitative analyses to share directly the voices of participants without modification. Fourth, deliberate balancing was performed to counteract unintentional "cherry picking" of only perspectives that fit a particular narrative. For example, in Chapters 3 and 5, quote counts were tallied throughout the analysis process to prevent talkative interviewees from dominating the results or potent but infrequently shared ideas from being missed. Overall, though the narrative presented in this dissertation regarding qualitative data is surely biased, we encourage future researchers to further explore and interrogate our data. Four ethical review processes were performed as part of this dissertation under the protocol numbers UVA-IRB-SBS-5614 (Phase 1 reviewed by University of Virginia IRB for Social and Behavioral Sciences and Phase 2 reviewed by UVA-SBS and Virginia State Uni-

versity IRB), and UVA-IRB-SBS-7557 (reviewed by UVA-SBS). Both protocols, 5614 and 7557, explicitly allow outside researchers to request permission from the investigators for anonymized copies of empirical data for further research purposes. This provides a final, ongoing source of trustworthiness for the honesty of our qualitative work [65]. To seek access to this data, please search for up-to-date institutional contacts for the author of this dissertation and his principal investigator, who is listed as his graduate advisor on the approval page in the front matter.

Statistical Validity

This dissertation made heavy use of inferential statistics (i.e., hypothesis testing) to support claims about the effects of GitHub Copilot and the associations of various measures of interest. Even if the measures and experimental design were valid, type I errors (i.e., false positive significance) or type II errors (i.e., false negative significance) are common. Our small samples make these concerns especially important, because parametric frequentist statistics rely on samples being sufficiently large to converge toward a correct answer even if the observations are not perfectly in line with the truth. Several approaches were used to counteract these challenges. In Chapter 2, which tested the effect of Copilot compared to a solo control and the effect of time (being on the second trial replicate rather than the first) within each condition, simple paired comparisons (aggregated when necessary) were made. Rather than naively using the paired t -test, which relies on a number of assumptions for valid inference, these assumptions were checked, and the least conservative reasonable test was used among a group of three increasingly conservative options. We hope this prevented type I errors, but it did risk producing type II errors by deflating p -values. For example, temporal workload—how rushed the participants felt—was lower by a non-statistically significant margin in Chapter 2. However, when including unpublished data collected in a similar manner from the participants in Chapter 5, that measure seems to cross the significance threshold, changing the overall conclusion. Different explanations, like differences between the cohorts and data collection circumstances might explain this, but sample size is another very plausible explanation. With more consequential measures like emotional valence and arousal, the conclusions did not seem to change. Nonetheless, it is always possible

that with large enough sample sizes, any extant effects can become recognized as significant effects, even if small or practically insignificant. In Chapters 4 and 7, we used more advanced inference techniques centered around linear regression: Generalized Estimating Equations [63] and Wild Cluster Bootstrapping [107]. These trade off simplicity and ease of justification with a greater balance between under- and over-rejection risks with respect to statistical hypotheses. As is the case with any analysis, it is possible to influence the outcome by choosing a method which leads to a result the researcher wishes to see. In the choice of these methods, we attempted to ensure that multiple reasonable approaches to solving the statistical problems we had all generally arrived at similar conclusions.

Another issue with statistical conclusions is over-reliance on p -values, especially without consideration of the multiple-comparison problem. The issue is that significance thresholds, such as our commonly used $\alpha = .05$, strive to reduce the incidence of type I error to 5%. This means that the expected number of wrong significant answers is one per 20 positive ($p < .05$) tests. Careful readers can revisit Chapters 2, 4 and 7 to estimate how many of the significant results discussed might reasonably be incorrect. The famous (or perhaps infamous) Bonferroni correction balloons p -values by a simple heuristic that then creates problems of type II error. A middle ground is a correction like Benjamini-Hochberg [7], which controls the False Discovery Rate at $(100\alpha)\%$ as intended. This approach was taken in Chapter 7. One might wonder why no p -value corrections were performed in Chapters 2 and 4. The answer comes down to a technicality about the nature of the questions being asked. If a researcher sets out hypotheses about a number of individual metrics in advance, as done in Chapters 2 and 4, it is reasonable not to adjust p -values unless a question can be answered by the significance of a singular sub-item. NASA-TLX workload is a perfect example because it is a six-item scale that attempts to measure one overarching construct—workload. In Chapter 2, our method corresponds to hypothesizing about *each* dimension on its own. We found that mental demand was lower with Copilot, for example, but did not use that evidence to conclude that physical demand or overall workload was lower. By contrast, in Chapter 7, we selected a subset of measures of interest and performed p -value corrections across that family of tests. This means that any one of our three significant findings could be

individually indicative of some difference in the aggregate workload represented by this group of four items. Although, we did not choose to make such a definitive conclusion. By contrast, if we had collected 100 random measures and observed a difference in a handful of them without any pre-made reasoning or expectations, it would then be inappropriate to interpret sub- α p -values as significant unless we had also performed a correction across all 100 tests. In summary, we attempted to balance the risks of type I and type II error with respect to inferential statistics, but other metrics like the central tendencies of the data or the calculated effect sizes can further elucidate the strength of the quantitative findings from Chapters 2, 4 and 7.

Endogeneity

Endogenous influences, those produced from within, threatened to influence our results through measurement error, omitted variables, and reverse causality. Sources of measurement error in our quantitative measures were inconsistencies between sets of tasks and variability in participants' understanding and interaction with subjective scales. Even if the task pools were somewhat exchangeable, such that there was no systematic difference in their difficulty, certain properties of tasks could have created noise in the data. For example, a programming task that requires knowledge of trigonometry might have greatly variable difficulty between individuals with different backgrounds in mathematics. Even if we were to know for sure that this was true, collecting a measure with which to control for this factor would be very hard if not impossible. Subjective scales are also inherently prone to measurement error. A participant can click down the center of a page of Likert bubbles mindlessly or even misread a question by accident. By instructing participants carefully and using validated or simple scales, we hope that statistical methods were able to wash out these sources of error. Omitted variables might include factors for the task pools or time. In Chapter 7 we specifically checked for endogeneity, but in Chapters 2 and 4, we relied instead on counterbalancing and statistical methods to overcome this source of error. For example, by distributing the assignment of task pools, orderings, and assignments of the treatment, any systematic differences in our metrics of interest owing to these factors should have been mixed up to where they would not produce spurious results. Reverse causality was probably

most concerning in Chapter 4, where associations between compliance and performance were statistically significant. While trust-to-compliance and performance-to-trust relationships could be thought of in causal terms based on the chronological ordering of measurement incidence, it is not entirely valid to conclude that compliance to Copilot's suggestions *caused* greater performance. An alternative explanation is that higher performers just happened to be more open to taking suggestions because, for example, they knew they could perform well either way. Though the laboratory studies in this dissertation took careful steps to counteract endogeneity, readers should still interpret our findings with these caveats in mind.

8.4.2 Threats to External Validity

The external validity of this work is the extent to which any internally valid findings generalize beyond our samples into contexts of practical importance. This work mostly aims to inform computing education, so this is the most important context in which to be valid. Practitioners or professional software developers may also draw from this dissertation, so industrial programming is also relevant.

Ecological Validity

A common criticism of the empirical work presented in this dissertation is that HumanEval [18] tasks are not generalizable to real-world educational or industrial programming tasks. Critics will point to their simplicity, well-defined requirements and acceptance criteria, and use for benchmarking LLMs. All these factors, we agree, make HumanEval tasks easier for GitHub Copilot than some theoretically "average" real-world task. Nonetheless, this criticism distracts from the real questions this dissertation answered about the performance benefits of Copilot. Whether Copilot was reliable for the tasks at hand was not of interest, but rather whether *novice and intermediate programmers* were capable of *harnessing* this reliability to their benefit—for performance, for learning, or for their general well-being. This goal will surely age poorly, as the obvious ease of obtaining such benefits will undoubtedly become even more clear in the years

to follow 2026. Let this note serve as a reminder for future scholars that 2022, when this work was first envisioned, was a much more uncertain time in this regard. As silly as it may sound, few could say confidently at that time whether or not novices were ready to use AI for programming.

Another fair criticism of Chapter 2, which is addressed therein, is the lack of internet access in the control condition, something that re-emerged in different forms after modifying subsequent study designs. Participants at VSU (see Chapter 5) were allowed to use the internet, but other unexpected controls ended up being needed for consistency there, too. For example, participants could not be permitted to message friends online or use ChatGPT in the browser, since our data collection system was unprepared for such choices. Some participants were even refused requests for scratch paper and permission to review their own nearby notebooks. By the time of Chapter 7, Google had begun including AI search overviews above nearly every set of returned internet search results. To restrict the use of AI to Copilot, therefore, this commonplace feature had to be disabled. Every one of these restrictions represents a tradeoff between internal validity and external validity. What serves to scientifically isolate a treatment risks also to diverge far enough from regular behavior as to make the resulting answers useless in practice. In a naturalistic study, programmers would be expected to exhibit wide variance in their help-seeking behaviors [43]. Only with a very large quantity of data could this additional noise be expected to allow treatment effects to show through appropriately.

Lastly, the incentive and risk structures in the solo versus AI-assisted design were potentially problematic. As noted particularly in Chapters 2 to 4 and 6, participants in the first study design—the 17 at UVA and 10 at VSU—were incentivized with money to perform as many tasks as they could by whatever means necessary. They were also given pre-written unit tests from HumanEval as acceptance criteria for their work. This meant that (1) they were perfectly equipped with feedback cycles to minimize the risk of using Copilot, and (2) they were actively encouraged not to focus on learning in the interest of saving valuable time. The discussion of Chapter 4 delves deeper into the importance of point (1) in relation to trust. As for point (2), this clearly biased participants toward using Copilot in maladaptive ways, which was a major finding of our work. This logic may seem circular or contrived, but, as discussed at length in Chapter 6, productivity is

an ecologically valid incentive for software developers in industry. Even in educational settings, students often take assessments under time pressure and work on programming projects with definitive deadlines. We argue that seeing how students acted when placed under such an incentive structure is an essential contribution to the overall knowledge base. However, to make better consideration of the tradeoff between immediate performance and learning retention, the second study design, introduced in Chapter 7, incentivized a balance between the two by lying to participants (ethically). We believe this strategy was effective; one participant's response after debriefing was, "that's genius," reflecting that the financial incentive drove him to prioritize learning exactly as intended. Overall, the studies in this dissertation created conditions where some of the most concerning tendencies were bound to arise, but that does not completely invalidate their generalizability.

Sampling

It is important to generalize findings cautiously when considering the small convenience samples taken for this work. The best mitigation to offer in this case is information about the participants that readers can use to situate the results against other groups of people. Because these studies attempted to collect rich and ambitious datasets, they were long and complex for participants to get involved with. They involved many planning steps and a large time commitment, restricting the likelihood that busy students, those with weak executive functioning skills, or those with limitations due to transportation or other factors would participate. The properties of the 49 total participants are discussed at greater length throughout the preceding chapters, but many of them were Asian and Black, with Hispanic/Latinx and indigenous participants being poorly represented. There was a reasonable representation of both male and female participants, but gender non-conforming participants were not common. At most one participant, who chose not to share their gender, may have been non-binary. Though several international students were recruited at VSU (see Chapter 5), they were by no means a representative sample of the world population. Overall, the sample of 39 UVA students and 10 VSU students reflects two public research universities in Virginia (U.S.A.) and the demographics of highly organized students who attend them.

8.4.3 Threats to Construct Validity

Construct validity is the extent to which our data effectively measured what we hoped to explore. For qualitative data, a threat to construct validity is the choice of questions, the wording of those questions, and the interviewing style. For more information about these choices, refer to the methodology of Chapters 3 and 5 to 7. For workload, our methodologies chose not to consider the six dimensions of the NASA-TLX as a single construct, but rather as six individual dimensions. This approach contrasts with the creators' intended use for the scale but requires fewer assumptions about the mathematical relationships between the items [36]. Emotion was measured with the Russell [95] valence versus arousal model using intuitive Likert scales. However, emotion can be described in many different ways, including categorically rather than dimensionally [92, 81, 47]. The covariance of emotional valence and arousal raises further questions about how many emotion dimensions participants can really distinguish. For example, one can feel strong positive valence or weak positive valence but then also feel a higher or lower level of arousal at either of those valence levels. Arguably, it is confusing to present different ways to quantify the intensity of emotions on the same reporting instrument. Greater attention to these psychometric questions is beyond the scope of this research. In Chapter 4, factor analysis and retest-reliability tests were performed specifically to scrutinize the construct validity of measures for trust. Finally, quantifying performance and learning is inherently challenging. Here, we considered only the ability to write code that passed unit tests with various levels of assistance. More complete measures of performance might consider code complexity, security vulnerabilities, or runtime speed. Measures for learning could account for conceptual understanding or the ability to explain the purpose or logic behind a given code snippet.

8.5 Future Work

In addition to several calls for future work throughout the chapters 2 to 7, future research should explore (1) how to make the Educational Recommendations contributed here practical to imple-

ment, (2) to what extent different AIDE guardrails are effective at promoting learning, and (3) how to combine the benefits of human-human pair programming and AI assistance.

Section 8.3 argues for three policy approaches to avoid and three to embrace for the responsible use of AI code generators in computer science education, providing concrete examples for clarity. However, educators are often plagued by limitations on their time and ability to make curricular changes. Many university professors minted over the past decade may have just recently confined the labor of teaching computer science to a reliable and repeatable recipe just as AI came into vogue. There is a natural desire to resist AI for the simple reason that revising pedagogical approaches is too hard. Rarely does a curriculum improve as a result of a singular sweeping redesign. Instead, incremental changes over time are realistic ways to move toward better education systems. Instructors are in dire need of resources to support the transition to the proposed *Charge-Challenge-Check* coursework approach. From ways to take attendance to easily proctored assessments to AI code critique assignments, free and open-source curricular options will be essential for ensuring that all computing students can reap the benefits of AI.

Section 1.4.4 introduced some of the early options showing how to restrict GenAI in ways designed to protect students. In the interest of making sure educational interventions are evidence-informed, it is crucial that guardrails be empirically evaluated for effectiveness. There is logic behind the hypothesis that code generation, for example, is a dangerous capability for student programmers aiming to learn. However, as argued in Sections 8.3.1 and 8.3.2, there are inequity challenges that can result from inappropriately imposed restrictions on commonplace AI technology. Even if guardrails are self-imposed by students, they should be supporting, rather than hindering learning. A classroom setting would be ideal to study guardrails. Researchers should evaluate not only code-generation guardrails but perhaps novel ones related to metacognition, problem decomposition, documentation, testing, or other factors.

Lastly, there is a need to study the effects of AI code generators in different human team formats. This dissertation compared the human-AI paradigm to the solo human paradigm and the human-human team paradigm, but future investigations should compare the human-human-AI paradigm to the human-human and human-AI paradigms. [68] took important first steps in this

direction, but it remains to be seen if the affective benefits of human teammates can coexist with the performance benefits of AIDEs, as well as where learning fits into that equation. Practitioners and educators alike will need informed recommendations for how to support an optimal balance between effective assistance and human control if AI is to be added to social programming teams. Controlled laboratory research will be integral to better understanding these hybrid-intelligence programming paradigms.

8.6 Closing Thoughts

On its surface, this dissertation appears to be about Artificial Intelligence, but it is really about humans. The 49 individuals who participated in this research put the *evidence* in *evidence-informed systems engineering* and gave meaning to the work. The key takeaway from this work is that the value of human connection is irreplaceable by Artificial Intelligence. In Chapter 2, Copilot failed to show an emotional benefit, despite its strong productivity support. Chapters 3 and 6 showed how novices were especially susceptible to Copilot dominating interactions, robbing them of the control and ownership over their work that they wanted. The results from Chapters 4 and 5 revealed how Copilot's assistance draws programmers in, encouraging them to trust it more and become distracted from fruitful social alternatives. However, the findings of Chapter 7 failed to show a learning benefit that matched Copilot's initial performance boost. More importantly, participants gained much stronger emotional benefits from human programming partners. In 2022, the robots were coming [29]. In 2023, the robots got here [86]. In 2026, the robots have not yet replaced us. However, if we want them to, they probably can soon. We can give Copilot a voice, a face, a body, a charming laugh, or anything else that brings it closer from artificiality to authenticity. This dissertation has begun to explain how, without these things, AI has much to offer student programmers, but pales in comparison to authentic human partners. As the next generation of designers considers this gap and how to close it, they ought to consider whether perhaps some roles are left best filled by us humans.

References

- [1] Andrew Abbott. 2007. *Chaos of disciplines*. (Nachdr. ed.). Univ. of Chicago Press, Chicago, Ill. 259 pp. isbn: 978-0-226-00101-2 978-0-226-00100-5.
- [2] Matin Amoozadeh, David Daniels, Daye Nam, Aayush Kumar, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Trust in generative AI among students: an exploratory study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024: The 55th ACM Technical Symposium on Computer Science Education. ACM, Portland OR USA, (Mar. 7, 2024), 67–73. isbn: 979-8-4007-0423-9. doi:10.1145/3626252.3630842.
- [3] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: how programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7, 85–111, OOPSLA1. Number: OOPSLA1.
- [4] Lecia J. Barker, Charlie McDowell, and Kimberly Kalahar. 2009. Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin*, 41, 1, (Mar. 4, 2009), 153–157. doi:10.1145/1539024.1508923.
- [5] Kathy Baxter, Catherine Courage, and Kelly Caine. 2015. *Understanding your users: a practical guide to user research methods*. Morgan Kaufmann. Retrieved Aug. 28, 2024 from <https://books.google.com/books?hl=en&lr=&id=l9-cBAAQBAJ&oi=fnd&pg=PP1&dq=Understanding+your+users:+A+practical+guide+to+user+research+methods&ots=dEIEZAXoRZ&sig=4ckI45L5qds4FShqQmSV-u-KsF0>.
- [6] Alvine B. Belle, Callum Sutherland, Opeyemi O. Adesina, Sègla Kpodjedo, Nathanael Ojong, and Lisa Cole. 2023. Bolstering the persistence of black students in undergraduate computer science programs: a systematic mapping study. *ACM Transactions on Computing Education*, 23, 4, (Dec. 31, 2023), 1–42. doi:10.1145/3617896.
- [7] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57, 1, 289–300. Number: 1.
- [8] Jens Bennedsen and Michael E. Caspersen. 2019. Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10, 2, (Apr. 25, 2019), 30–36. doi:10.1145/3324888.
- [9] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. 2008. Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. *ACM SIGCSE Bulletin*, 40, 1, (Mar. 12, 2008), 402–406. doi:10.1145/1352322.1352274.

- [10] 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. *Commun. ACM*, 66, 6, (May 24, 2023), 56–62. doi:10.1145/3589996.
- [11] Margaret M. Blair and Steven M. H. Wallman. 2001. *Unseen wealth: report of the Brookings Task Force on Intangibles*. In collab. with Brookings Task Force on Intangibles. Brookings Institution Press, Washington, D.C. 1 p. isbn: 978-0-8157-0113-2 978-0-8157-9125-6.
- [12] Thierry Blayac, Dimitri Dubois, Sébastien Duchêne, Phu Nguyen-Van, Bruno Ventelou, and Marc Willinger. 2022. What drives the acceptability of restrictive health policies: an experimental assessment of individual preferences for anti-COVID 19 strategies. *Economic Modelling*, 116, 106047. Retrieved Mar. 11, 2026 from <https://www.sciencedirect.com/science/article/pii/S026499932200284X>.
- [13] Pierre Bourdieu. 1986. The forms of capital. In *The sociology of economic life*. (Third edition ed.). Mark S. Granovetter and Richard Swedberg, (Eds.) Routledge, Taylor & Francis Group, New York London. isbn: 978-0-429-49433-8.
- [14] Virginia Braun and Victoria Clarke. 2022. Conceptual and design thinking for thematic analysis. *Qualitative psychology*, 9, 1, 3. Retrieved July 12, 2024 from <https://psycnet.apa.org/fulltext/2021-45248-001.html>.
- [15] Wendy Brown. 2017. *Undoing the demos: neoliberalism's stealth revolution*. (First paperback edition, this edition corrects errors in previous printings ed.). *Near futures*. Zone Books, New York. 295 pp. isbn: 978-1-935408-54-3.
- [16] U.S. Census Bureau. 2024. QuickFacts: age and sex. U.S. Census Bureau: QuickFacts. Retrieved Mar. 18, 2025 from <https://www.census.gov/quickfacts/fact/table/US/PST045224>.
- [17] U.S. Census Bureau. 2024. QuickFacts: race and hispanic origin. U.S. Census Bureau: QuickFacts. Retrieved Mar. 18, 2025 from <https://www.census.gov/quickfacts/fact/table/US/PST045224>.
- [18] Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- [19] Jody Clarke-Midura, Frederick J. Poole, Katarina Pantic, Chongning Sun, and Vicki Allan. 2018. How mother and father support affect youths' interest in computer science. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ICER '18: International Computing Education Research Conference. ACM, Espoo Finland, (Aug. 8, 2018), 215–222. isbn: 978-1-4503-5628-2. doi:10.1145/3230977.3231003.
- [20] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub copilot AI pair programmer: asset or liability? *Journal of Systems and Software*, 203, (Sept. 1, 2023), 111734. doi:10.1016/j.jss.2023.111734.

- [21] Mark Daly. 2002. *Task Load and Automation Use in an Uncertain Environment*. Ph.D. Dissertation. (Mar. 1, 2002). <https://scholar.afit.edu/etd/4394>.
- [22] Cristobal de Brey, Kristi Donaldson, NCES Supervisors, and NCES Annual Reports {and} Information Staff. 2024. 2022 Digest of Education Statistics. Institute of Education Sciences- United States National Center for Education Statistics, Washington, DC, USA, (Feb. 2024). Retrieved Mar. 18, 2024 from <https://nces.ed.gov/programs/digest/d22/>.
- [23] Jennifer Dempsey, Richard T. Snodgrass, Isabel Kishi, and Allison Titcomb. 2015. The emerging role of self-perception in student intentions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. SIGCSE '15: The 46th ACM Technical Symposium on Computer Science Education. ACM, Kansas City Missouri USA, (Feb. 24, 2015), 108–113. isbn: 978-1-4503-2966-8. doi:10.1145/2676723.2677305.
- [24] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt problems: a new programming exercise for the generative AI era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 296–302. isbn: 979-8-4007-0423-9. doi:10.1145/3626252.3630909.
- [25] Amanda B. Diekman, Mia Steinberg, Elizabeth R. Brown, Aimee L. Belanger, and Emily K. Clark. 2017. A goal congruity model of role entry, engagement, and exit: understanding communal goal processes in STEM gender gaps. *Personality and Social Psychology Review*, 21, 2, (May 2017), 142–175. doi:10.1177/1088868316642141.
- [26] Ron Eglash, Audrey Bennett, Laquana Cooke, William Babbitt, and Michael Lachney. 2021. Counter-hegemonic computing: toward computer science education for value generation and emancipation. *ACM Trans. Comput. Educ.*, 21, 4, (Oct. 15, 2021), 29:1–29:30. doi:10.1145/3449024.
- [27] Guangrui Fan, Dandan Liu, Rui Zhang, and Lihu Pan. 2025. The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, 12, 1, (Mar. 4, 2025), 16. doi:10.1186/s40594-025-00537-3.
- [28] John Fantuzzo. 2015. A course between bureaucracy and charisma: a pedagogical reading of max weber's social theory. *Journal of Philosophy of Education*, 49, 1, (Feb. 2015), 45–64. doi:10.1111/1467-9752.12095.
- [29] 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 (ACE '22). Association for Computing Machinery, New York, NY, USA, (Feb. 14, 2022), 10–19. isbn: 978-1-4503-9643-1. doi:10.1145/3511861.3511863.
- [30] 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 (ACE '23)*. Association for Computing Machinery, New York, NY, USA, (Jan. 30, 2023), 97–104. isbn: 978-1-4503-9941-8. doi:10.1145/3576123.3576134.
- [31] Gian Luca Scoccia. 2023. Exploring early adopters' perceptions of ChatGPT as a code generation tool, (Sept. 11, 2023). MAG ID: 4388214696. doi:10.1109/asew60602.2023.00016.
- [32] Henry A. Giroux. 2019. *Neoliberalism's War on Higher Education*. (2nd ed ed.). Haymarket Books, La Vergne. 1 p. isbn: 978-1-60846-334-3 978-1-64259-092-0.
- [33] GitHub. 2023. GitHub copilot · your AI pair programmer. GitHub. Retrieved July 24, 2023 from <https://github.com/features/copilot>.
- [34] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. Pair programming in education: a literature review. *Computer Science Education*, 21, 2, 135–173.
- [35] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I. K. Sjøberg. 2009. The effectiveness of pair programming: a meta-analysis. *Information and Software Technology*. Special Section: Software Engineering for Secure Systems 51, 7, (July 1, 2009), 1110–1122. doi:10.1016/j.infsof.2009.02.001.
- [36] Sandra G. Hart. 2006. Nasa-task load index (NASA-TLX); 20 years later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 50, 9, (Oct. 2006), 904–908. doi:10.1177/154193120605000909.
- [37] David Hartley. 1995. The 'McDonaldization' of higher education: food for thought? *Oxford Review of Education*, 21, 4, 409–423. Retrieved July 9, 2025 from <https://www.jstor.org/stable/1050719>.
- [38] Jonathan E. Haskel and Stian Westlake. 2018. *Capitalism without capital: the rise of the intangible economy*. Princeton university press, Princeton. isbn: 978-1-4008-8832-0 978-0-691-17503-4.
- [39] Anja Hawlitschek, Sarah Berndt, and Sandra Schulz. 2023. Empirical research on pair programming in higher education: a literature review. *Computer Science Education*, 33, 3, (July 3, 2023), 400–428. doi:10.1080/08993408.2022.2039504.
- [40] Kevin Anthony Hoff and Masooda Bashir. 2015. Trust in automation: integrating empirical evidence on factors that influence trust. *Human Factors*, 57, 3, (May 1, 2015), 407–434. doi:10.1177/0018720814547570.
- [41] John L. Holland. 1985. *Making Vocational Choices: A Theory of Vocational Personalities and Work Environments*. Prentice-Hall, Englewood Cliff, NJ. isbn: 978-0-13-547597-3.

- [42] Diane Horton and Michelle Craig. 2015. Drop, fail, pass, continue: persistence in CS1 and beyond in traditional and inverted delivery. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. SIGCSE '15: The 46th ACM Technical Symposium on Computer Science Education. ACM, Kansas City Missouri USA, (Feb. 24, 2015), 235–240. isbn: 978-1-4503-2966-8. doi:10.1145/2676723.2677273.
- [43] Irene Hou, Sophia Mettillle, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The effects of generative AI on computing students' help-seeking preferences. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*. Association for Computing Machinery, New York, NY, USA, (Jan. 29, 2024), 39–48. isbn: 979-8-4007-1619-5. doi:10.1145/3636243.3636248.
- [44] Anna Y. Q. Huang, Cheng-Yan Lin, Sheng-Yi Su, and Stephen J. H. Yang. 2025. The impact of GenAI-enabled coding hints on students' programming performance and cognitive load in an SRL-based python course. *British Journal of Educational Technology*, 56, 5, 1942–1972. _eprint: <https://bera-journals.onlinelibrary.wiley.com/doi/pdf/10.1111/bjet.13589>. doi:10.1111/bjet.13589.
- [45] Walter Humes. 2022. THE 'IRON CAGE' OF EDUCATIONAL BUREAUCRACY. *British Journal of Educational Studies*, 70, 2, (Mar. 4, 2022), 235–253. doi:10.1080/00071005.2021.1899129.
- [46] Yujeong Hwang, Anjali Das, Jane Waite, and Sue Sentence. 2023. Using a sociological lens to investigate computing teachers' culturally responsive classroom practices. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23)*. Vol. 1. Association for Computing Machinery, New York, NY, USA, (Sept. 10, 2023), 206–221. isbn: 978-1-4503-9976-0. doi:10.1145/3568813.3600112.
- [47] IEEE. 2020. IEEE recommended practice for assessing the impact of autonomous and intelligent systems on human well-being. *IEEE Std 7010-2020*, (May 2020), 1–96. Conference Name: IEEE Std 7010-2020. doi:10.1109/IEEESTD.2020.9084219.
- [48] Saki Imai. 2022. Is github copilot a substitute for human pair-programming? an empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 319–321.
- [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*, 1–15.
- [50] Tony Jenkins. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* number 2002. Vol. 4. Citeseer, 53–58. Retrieved

Nov. 7, 2023 from <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ed6b9c9da3dde0f884351f98155b051ec371dc3>.

- [51] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J. Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–19.
- [52] Maria Kallia. 2025. To be, or to be otherwise? silicon souls in search of dasein and authentic human engagement in the age of generative AI. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1 (ICER '25)*. Association for Computing Machinery, New York, NY, USA, (Aug. 2, 2025), 240–255. isbn: 979-8-4007-1340-8. doi:10.1145/3702652.3744223.
- [53] Maria Kallia and Quintin Cutts. 2021. Re-examining inequalities in computer science participation from a bourdieusian sociological perspective. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021)*. Association for Computing Machinery, New York, NY, USA, (Aug. 17, 2021), 379–392. isbn: 978-1-4503-8326-4. doi:10.1145/3446871.3469763.
- [54] 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, (Apr. 19, 2023), 1–23. isbn: 978-1-4503-9421-5. doi:10.1145/3544548.3580919.
- [55] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–12. isbn: 979-8-4007-1653-9. doi:10.1145/3631802.3631806.
- [56] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, (May 11, 2024), 1–20. isbn: 979-8-4007-0330-0. doi:10.1145/3613904.3642773.
- [57] Amy J. Ko, Francisco Castro, Aakash Gautam, Anne Drew Hu, Sara Kingsley, Michael Lachney, and Aman Yadav. 2024. Computing, education, and capitalism. In *Proceedings of the 55th ACM Technical*

- Symposium on Computer Science Education V. 2* (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, (Mar. 15, 2024), 1914. isbn: 979-8-4007-0424-6. doi:10.1145/3626253.3635362.
- [58] Sandeep Kaur Kuttal, Bali Ong, Kate Kwasny, and Peter Robe. 2021. Trade-offs for substituting a human with an agent in a pair programming context: the good, the bad, and the ugly. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (CHI '21). Association for Computing Machinery, New York, NY, USA, (May 7, 2021), 1–20. isbn: 978-1-4503-8096-6. doi:10.1145/3411764.3445659.
- [59] Michael Lachney, Jean Ryoo, and Rafi Santo. 2021. Introduction to the special section on justice-centered computing education, part 1. *ACM Trans. Comput. Educ.*, 21, 4, (Dec. 6, 2021), 25:1–25:15. doi:10.1145/3477981.
- [60] Sam Lau and Philip 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 2023: ACM Conference on International Computing Education Research. ACM, Chicago IL USA, (Aug. 7, 2023), 106–121. isbn: 978-1-4503-9976-0. doi:10.1145/3568813.3600138.
- [61] John D. Lee and Katrina A. See. 2004. Trust in automation: designing for appropriate reliance. *Human factors*, 46, 1, 50–80. Publisher: SAGE Publications Sage UK: London, England.
- [62] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A large-scale survey on the usability of AI programming assistants: successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (ICSE '24). Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3608128.
- [63] Kung-Yee Liang and Scott L. Zeger. 1986. Longitudinal data analysis using generalized linear models. *Biometrika*, 73, 1, 13–22. doi:10.2307/2336267.
- [64] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. Koli Calling '23: 23rd Koli Calling International Conference on Computing Education Research. ACM, Koli Finland, (Nov. 13, 2023), 1–11. isbn: 979-8-4007-1653-9. doi:10.1145/3631802.3631830.
- [65] Yvonna S. Lincoln and Egon G. Guba. 1985. *Naturalistic inquiry*. Vol. 75. Sage. Retrieved Aug. 28, 2024 from <https://books.google.com/books?hl=en&lr=&id=2oA9aWlNeooC&oi=fnd&pg=PA7&dq=lincoln+and+guba+naturalistic&ots=Ov1xYcVfvm&sig=0g8kKlMXyrx54UVh08HGQdrXLZI>.

- [66] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024. Teaching CS50 with AI: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, (Mar. 7, 2024), 750–756. isbn: 979-8-4007-0423-9. doi:10.1145/3626252.3630938.
- [67] Andrew Luxton-Reilly. 2016. Learning to program is easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '16: Innovation and Technology in Computer Science Education Conference 2016. ACM, Arequipa Peru, (July 11, 2016), 284–289. isbn: 978-1-4503-4231-5. doi:10.1145/2899415.2899432.
- [68] Wenhan Lyu, Yimeng Wang, Yifan Sun, and Yixuan Zhang. 2025. Will your next pair programming partner be human? an empirical evaluation of generative AI as a collaborative teammate in a semester-long classroom setting. In *Proceedings of the Twelfth ACM Conference on Learning @ Scale*. L@S '25: Twelfth ACM Conference on Learning @ Scale. ACM, Palermo Italy, (July 17, 2025), 83–94. isbn: 979-8-4007-1291-3. doi:10.1145/3698205.3729544.
- [69] Qianou Ma, Tongshuang Wu, and Kenneth Koedinger. 2023. Is AI the better programming partner? human-human pair programming vs. human-AI pAIr programming. (June 9, 2023). arXiv: 2306.05153[cs]. doi:10.48550/arXiv.2306.05153.
- [70] Lauri Malmi et al. 2014. Theoretical underpinnings of computing education research: what is the evidence? In *Proceedings of the tenth annual conference on International computing education research* (ICER '14). Association for Computing Machinery, New York, NY, USA, (July 28, 2014), 27–34. isbn: 978-1-4503-2755-8. doi:10.1145/2632320.2632358.
- [71] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-regulation, self-efficacy, and fear of failure interactions with how novices use LLMs to solve programming problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2024: Innovation and Technology in Computer Science Education. ACM, Milan Italy, (July 3, 2024), 276–282. isbn: 979-8-4007-0600-4. doi:10.1145/3649217.3653621.
- [72] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. 2002. The effects of pair-programming on performance in an introductory programming course. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 38–42.

- [73] John M. McGuirl and Nadine B. Sarter. 2006. Supporting trust calibration and the effective use of decision aids by presenting dynamic system confidence information. *Human Factors*, 48, 4, (Dec. 1, 2006), 656–665. doi:10.1518/001872006779166334.
- [74] Stephanie M. Merritt and Daniel R. Ilgen. 2008. Not all trust is created equal: dispositional and history-based trust in human-automation interactions. *Human factors*, 50, 2, 194–210. Number: 2 Publisher: SAGE Publications Sage CA: Los Angeles, CA.
- [75] Bonnie M. Muir. 1989. *Operators' trust in and use of automatic controllers in a supervisory process control task*. Ph.D. Dissertation. University of Toronto (Canada), Canada – Ontario, CA. isbn: 978-0-315-51014-2. Retrieved June 29, 2023 from <https://www.proquest.com/pqdtglobal/docview/303794539/citation/EA1C186F755147DBPQ/1>.
- [76] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24: IEEE/ACM 46th International Conference on Software Engineering. ACM, Lisbon Portugal, (Apr. 12, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3639187.
- [77] National Institute of Standards and Technology (NIST). 2004. CVE-2004-0113 detail. (Mar. 3, 2004). Retrieved Mar. 12, 2026 from <https://nvd.nist.gov/vuln/detail/CVE-2004-0113>.
- [78] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22)*. Association for Computing Machinery, New York, NY, USA, (Oct. 17, 2022), 1–5. isbn: 978-1-4503-9303-4. doi:10.1145/3524842.3528470.
- [79] Colleen E. Patton and Christopher D. Wickens. 2024. The relationship of trust and dependence. *Ergonomics*, 67, 11, (Nov. 2024), 1535–1552. doi:10.1080/00140139.2024.2342436.
- [80] 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.
- [81] Reinhard Pekrun. 2024. Control-value theory: from achievement emotion to a general theory of human emotions. *Educational Psychology Review*, 36, 3, (Sept. 2024), 83. doi:10.1007/s10648-024-09909-7.
- [82] Jacob Penney, Pawan Acharya, Peter Hilbert, Priyanka Parekh, Anita Sarma, Igor Steinmacher, and Marco Gerosa. 2025. Understanding programming students' help-seeking preferences in the era of generative AI. In *Proceedings of the ACM Global Computing Education Conference 2025 - Volume 1*. CompEd 2025: ACM Global Computing Education Conference 2025. ACM, Gaborone Botswana, (Oct. 21, 2025), 15–21. isbn: 979-8-4007-1929-5. doi:10.1145/3736181.3747165.

- [83] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, (Nov. 21, 2023), 2785–2799. isbn: 979-8-4007-0050-7. doi:10.1145/3576915.3623157.
- [84] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: the benefits and harms of generative AI for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1. ICER 2024: ACM Conference on International Computing Education Research*. ACM, Melbourne VIC Australia, (Aug. 12, 2024), 469–486. isbn: 979-8-4007-0475-8. doi:10.1145/3632620.3671116.
- [85] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. “it’s weird that it knows what i want”: usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 29, 2024), 1–31. doi:10.1145/3617367.
- [86] James Prather et al. 2023. The robots are here: navigating the generative AI revolution in computing education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITICSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, (Dec. 28, 2023), 108–159. isbn: 979-8-4007-0405-5. doi:10.1145/3623762.3633499.
- [87] Yolanda A. Rankin and Jakita O. Thomas. 2020. The intersectional experiences of black women in computing. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, (Feb. 26, 2020), 199–205. isbn: 978-1-4503-6793-6. doi:10.1145/3328778.3366873.
- [88] Yolanda A. Rankin, Jakita O. Thomas, and Sheena Erete. 2021. Black women speak: examining power, privilege, and identity in CS education. *ACM Transactions on Computing Education*, 21, 4, (Oct. 25, 2021), 26:1–26:31. doi:10.1145/3451344.
- [89] Lauren Rich, Heather Perry, and Mark Guzdial. 2004. A CS1 course designed to address interests of women. *ACM SIGCSE Bulletin*, 36, 1, (Mar. 2004), 190–194. doi:10.1145/1028174.971370.
- [90] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: a review and discussion. *Computer Science Education*, 13, 2, (June 1, 2003), 137–172. _eprint: <https://doi.org/10.1076/csed.13.2.137.14200>. doi:10.1076/csed.13.2.137.14200.

- [91] Lucero Rodriguez Rodriguez, Carlos E. Bustamante Orellana, Erin K. Chiou, Lixiao Huang, Nancy Cooke, and Yun Kang. 2023. A review of mathematical models of human trust in automation. *Frontiers in Neuroergonomics*, 4, 1171403. Retrieved Jan. 30, 2026 from <https://www.frontiersin.org/journals/neuroergonomics/articles/10.3389/fnrgo.2023.1171403/full>.
- [92] Ira J. Roseman. 1996. Appraisal determinants of emotions: constructing a more accurate and comprehensive theory. *Cognition & Emotion*, 10, 3, (May 1996), 241–278. doi:10.1080/026999396380240.
- [93] 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. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. IUI '23: 28th International Conference on Intelligent User Interfaces. ACM, Sydney NSW Australia, (Mar. 27, 2023), 491–514. isbn: 979-8-4007-0106-1. doi:10.1145/3581641.3584037.
- [94] Emma Rowe. 2019. Capitalism without capital: the intangible economy of education reform. *Discourse: Studies in the Cultural Politics of Education*, 40, 2, (Mar. 4, 2019), 271–279. doi:10.1080/01596306.2019.1569883.
- [95] James A. Russell. 1980. A circumplex model of affect. *Journal of personality and social psychology*, 39, 6, 1161. Retrieved Nov. 7, 2023 from <https://psycnet.apa.org/journals/psp/39/6/1161/>.
- [96] Jean Salac, Lena Armstrong, F. Megumi Kivuva, Jayne Everson, and Amy J. Ko. 2024. How economically marginalized adolescents of color negotiate critical pedagogy in a computing classroom. *ACM Trans. Comput. Educ.*, 25, 1, (Dec. 17, 2024), 2:1–2:29. doi:10.1145/3702330.
- [97] Norsaremah Salleh, Emilia Mendes, and John Grundy. 2011. Empirical studies of pair programming for CS/SE teaching in higher education: a systematic literature review. *IEEE Transactions on Software Engineering*, 37, 4, (July 2011), 509–525. doi:10.1109/TSE.2010.59.
- [98] Anshul Shah, Anya Chernova, Elena Tomson, Leo Porter, William G. Griswold, and Adalbert Gerald Soosai Raj. 2025. Students’ use of GitHub copilot for working with large code bases. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, (Feb. 18, 2025), 1050–1056. isbn: 979-8-4007-0531-1. doi:10.1145/3641554.3701800.
- [99] Suzanne StolzSuzanne Stolz. 2021. Universal design for learning: changing the way we interact with diversity. In *The Oxford Encyclopedia of Inclusive and Special Education*. Oxford University Press, (May 20, 2021). isbn: 978-0-19-087587-9. Retrieved Mar. 12, 2026 from <https://www.oxfordreference.com/display/10.1093/acref/9780190875879.001.0001/acref-9780190875879-e-1020>.

- [100] 2014. *The culturally engaging campus environments (CECE) model: a new theory of success among racially diverse college student populations*. Samuel D. Museus. *Higher Education: Handbook of Theory and Research*. Springer Netherlands, Dordrecht, 189–227. isbn: 978-94-017-8004-9 978-94-017-8005-6. doi:10.1007/978-94-017-8005-6_5.
- [101] E. L. Trist and K. W. Bamforth. 1951. Some social and psychological consequences of the longwall method of coal-getting: an examination of the psychological situation and defences of a work group in relation to the social structure and technological content of the work system. *Human Relations*, 4, 1, (Feb. 1951), 3–38. doi:10.1177/001872675100400101.
- [102] U.S. Dept. of Commerce, National Telecommunications and Information Administration. 1999. Falling through the net: defining the digital divide. Retrieved Feb. 27, 2025 from http://security.cc.columbia.edu/itc/polisci/W3923/digital_divide.pdf.
- [103] Karthikeyan Umapathy and Albert D. Ritzhaupt. 2017. A meta-analysis of pair-programming in computer programming courses: implications for educational practice. *ACM Transactions on Computing Education (TOCE)*, 17, 4, 1–13.
- [104] 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 (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, (Apr. 28, 2022), 1–7. isbn: 978-1-4503-9156-6. doi:10.1145/3491101.3519665.
- [105] Marcel Valový and Alena Buchalceva. 2023. The psychological effects of AI-assisted programming on students and professionals. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME). ISSN: 2576-3148. (Oct. 2023), 385–390. doi:10.1109/ICSME58846.2023.00050.
- [106] Tina Vrieler and Minna Salminen-Karlsson. 2022. A sociocultural perspective on computer science capital and its pedagogical implications in computer science education. *ACM Trans. Comput. Educ.*, 22, 4, (Sept. 15, 2022), 44:1–44:23. doi:10.1145/3487052.
- [107] Matthew D. Webb. 2023. Reworking wild bootstrap-based inference for clustered errors. *Canadian Journal of Economics/Revue canadienne d'économique*, 56, 3, (Aug. 2023), 839–858. doi:10.1111/caje.12661.
- [108] Justin D. Weisz et al. 2021. Perfection not required? human-AI partnerships in code translation. In MAG ID: 3154248444. (Apr. 14, 2021), 402–412. doi:10.1145/3397481.3450656.
- [109] Jacob M. Wetzel, Sonya M. Sheffert, and Richard W. Backs. 2004. Driver trust, annoyance, and acceptance of an automated calendar system. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 48, 19, (Sept. 1, 2004), 2335–2339. doi:10.1177/154193120404801928.

- [110] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE code generation from natural language: promise and challenges. *ACM Transactions on Software Engineering and Methodology*, 31, 2, (Mar. 4, 2022), 29:1–29:47. doi:10.1145/3487569.
- [111] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, 62–71.
- [112] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, 4, (Jan. 1, 2023), 100147. doi:10.1016/j.caeai.2023.100147.
- [113] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in computing education: perspectives of students and instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*. 2023 IEEE Frontiers in Education Conference (FIE). (Oct. 2023), 1–9. doi:10.1109/FIE58773.2023.10343467.

Appendix A

Interview Guide for Chapter 3

1. Which features did you use?
2. Did you use some more than others? Why?
3. Were you able to find a good rhythm working with Copilot? If so, how long did it take to get there?
4. Did you have any strategies you used to work more effectively with Copilot's capabilities?
5. Did your process for solving a problem differ when you had Copilot as compared to when you didn't. If so, how?
6. Did you notice any differences in your feelings or attitude when you worked with Copilot as compared to working alone? Can you elaborate?
7. Did Copilot ever get in your way or make it harder to program? Can you tell me about those times?
8. Did Copilot ever get you "unstuck" or help you move forward? Can you share an example?
9. How difficult or easy were Copilot's suggestions to understand? Were any much easier or more difficult?

10. How do you think regular use of Copilot for programming would affect your learning? Would you learn more, less, or the same amount? What about the pace of learning?
11. Do you think any experiences from today will impact how you program in the future? Can you elaborate on that?
12. How did your experiences today affect your confidence in your ability to succeed in your degree program?
13. How do you think this technology will impact innovation?
14. Do you have any concerns or fears about a product like this in your industry?
15. What is your overall impression of this technology?

Appendix B

Qualitative Codebook for Chapter 3

B.1 Interaction Patterns

Both the human and the AI drive various phenomena to occur within or resulting from their interactions.

B.1.1 Acceleration

Programmers use AI to work more quickly and to maintain flow.

B.1.2 Shepherding

Programmers apply deliberate techniques to coerce AI to help them reach their goals.

- *adapting*: Programmers can modify AI generated code to better meet their needs.
- *backtracking*: Programmers can delete some or all of the code and comments in their environment when the AI is not helping them appropriately with what they have.
- *decomposition*: Programmers can break down problems into smaller sub-tasks prior to seeking AI assistance.
- *drifting*: Programmers may waste time with repeated, fruitless attempts to shepherd AI.

- *prompting*: Programmers can communicate with AI using text.
 - *judicious*: Users can intentionally provide concise prompts to yield better AI suggestions.
 - *with code*: Programmers can write code with the hope that it helps the AI produce better suggestions.
 - *with comments*: In early versions of GitHub Copilot, comments were the only way to provide natural language prompts.
 - *incidental*: Context for the AI is left behind automatically as a programmer works within an AIDE.
- *slow accepting*: Users can type out AI-generated suggestions character-by-character in lieu of accepting them wholesale.
- *acquired skill*: Shepherding AI is a skill at which programmers can become better through experience.

B.1.3 Mindlessness

Novice programmers focus on actions and cues in the programming environment, without taking time to plan or think through problems deliberately.

- *tunnel vision*: Focusing on what is salient and immediate causes novice programmers to overlook useful resources and forgo thinking ahead.
- *rushing*: Time pressure exacerbates novices' mindless tendencies because immediate progress seems important for reaching their goals quickly.
- *forgetting*: Users fail to use features they don't remember.
- *myopia*: Novices can focus on the near-term, failing to think ahead or look at the bigger picture.

B.1.4 Influencing

The availability of generative AI affects programmers' thoughts and actions while they work.

- *distracting*: Cues from AI can lead programmers to lose focus and abandon or modify their own plans.
 - *polluting*: Cues from AI can give users new information or ideas that they are unable to forget or ignore.
 - *tempting*: The prospect of assistance from the AI can override a programmer's desire to engage with a task fully or independently.
 - *rerouting*: AI can cause programmers to take a different approach than they would have alone.
- *variety*: AI exposes programmers to differential approaches, new ideas, and diverse programming constructs without their having to seek them out intentionally.
 - *branching*: With one starting point, there are many different directions that AI can lead a programmer.
 - *courage*: Seeing AI using an approach a programmer is less familiar with can impart courage to try the approach, since the AI will be there for support.
- *style*: AI affects the conventions, structure, and documentation a programmer writes or leaves in code.
- *teaching*: AI produces content that can teach users new ideas and help them learn better than they would alone.

B.1.5 Exploration

Programmers use AI to find new ideas and approaches and to move forward when they reach an impasse.

- *consulting*: Programmers sometimes view and read AI-generated code for informational purposes without necessarily intending to use it.
- *getting unstuck*: AI helps programmers get unstuck when they reach an impasse.
- *getting started*: AI helps programmers to overcome writer's block when they first start working on a task.
- *panel completions*: Early versions of GitHub Copilot facilitated exploration primarily through a manually triggered panel containing multiple code completion options for users to compare before use.

B.1.6 Mirroring

AI generated content often reflects attributes of the context provided by the programmer, such as code correctness or level of sophistication.

B.2 Interaction Quality

What happens during an interaction can leave positive and negative impressions on the human.

B.2.1 Dissonance

Interactions can cause negative reactions or feelings toward AI.

- *emotional*: Interactions with AI can cause users to have negative feelings.
 - *shame*: If the AI does too much of the work, users can feel guilty that they are not contributing enough.
 - *emptiness*: Using AI-assistance can prevent feelings of satisfaction that might otherwise accompany the completion of a task.

- *misalignment*: AI can fail to provide assistance that matches the needs and wants of the user.
 - *unhelpful*: AI can generate content that does not assist a user in reaching their goals.
 - *incorrect*: AI can generate code that does fails to run or meet functional requirements.
 - *enabling*: AI does not generally prevent users from writing incorrect code themselves.
 - *overload*: Large volumes of AI-generated can be overwhelming to a user's limited cognitive resources.
 - *confusion*: Users can struggle to make sense of AI behavior and AI-generated content.
 - * *unfamiliar*: AI can employ code constructs with which users are not familiar.
- *apprehension*: Users can feel nervous about the future after experiencing the impressive utility of AI.
- *mirage*: Users tolerate incorrect code more poorly when it appears correct.

B.2.2 Harmony

Users value and enjoy positive interactions with AI.

- *emotional*: Interactions with AI can cause users to have positive feelings.
- *alignment*: AI-generated content often matches the needs and wants of the user.
 - *clarity*: AI-generated content can be easy for users to understand.
 - *easier*: Working with AI can be easier than working without it.
 - *helpful*: AI can provide valuable assistance to users.

B.3 Work Allocation

Users can choose to use the AI or do work themselves.

B.3.1 Insourcing

Preference and necessity drive users to do certain aspects of work themselves.

- *control*: A desire to maintain authority within an AI-interaction can drive users to do work themselves.
- *creativity*: Humans contribute inventiveness and originality to interactions with AI.
- *slow rejecting*: A user can implicitly reject an AI-generated inline suggestion by typing something other than what is suggested.
- *evaluating*: Users have to assess the quality of AI-generated content to determine whether it is useful.
 - *inspection*: Reading and interpreting AI-generated code helps users understand it and predict how it will work.
 - *pattern matching*: Some users skim AI-generated code looking for properties they expect a good suggestion might have.
 - *tinkering*: Cyclically changing and re-evaluating code helps users increase their knowledge about it and its correctness.
 - ★ *debugging*: Users work cyclically to identify and eliminate functional issues with code.
 - *executing*: Running code allows programmers to assess its behavior and functional correctness.
 - ★ *testing*: Programmers can run testing code to evaluate other code.
- *understanding*: Programmers who allow AI to solve a problem for them often still desire to examine solution and know how it works.
- *fun*: It can be more enjoyable do work oneself rather than having AI do it.

- *under-reliance*: Users have to do more work when they fail to take full advantage of what AI has to offer.
- *goal setting*: Users must decide what they want from AI in order to use it successfully.
- *problem solving*: Some users want to choose and architect the high-level approach for their code themselves rather than having an AIDE do so.

B.3.2 Outsourcing

Users can offload work to AI.

- *over-reliance*: Excessive use of AI assistance can disengage a human from the work required to produce programs.
 - *not learning*: When a programmer over-relies on AI assistance, they may not be prepared to do similar work alone in the future.
 - *not understanding*: Using AI-generated code without trying to know how it works may be considered a form of over-reliance.
- *simple work*: Programmers may outsource both the logic and expression of rudimentary programming tasks.
- *expression*: AI can translate programmers' ideas and plans into programming languages.
 - *syntax*: AI can help programmers write code that adheres to the grammar of a programming language.
 - *typing*: AI can help programmers by writing out code they would otherwise write out manually.

B.4 Appraisals

Users measure their own capabilities and those of the AI while assigning value to certain interaction behaviors.

B.4.1 Responsibility

Righteous use of AI demands restriction to certain conditions and considerations of various moral contingencies.

- *mindset*: Responsible interactions occur when immediate intentions align with a higher moral imperative.
- *integrity*: Internal choices to adhere to standards are what create scrupulousness in AI interactions.
- *accountability*: A programmer proffers not only a software deliverable itself, but also a degree of liability to answer for its quality and the opinionation of its design.
- *moderation*: It is possible to use AI too much.
- *maturity*: Young and inexperienced programmers are least able to use AI properly.
- *context*: The moral value of an action can differ based on when, where, or how it is done.
- *participation*: Responsible use of AI should and often must involve human involvement in work process.
 - *desiring*: Users want to be involved in work with which AI is helping.
 - *required*: AI is often not able to function autonomously, so a human must be involved for it to fulfill its intended purpose.

B.4.2 Self-efficacy

Feelings about one's own abilities and worth stem from both the outcomes of programming efforts and the means used to achieve them.

- *high*: Users feel more confident in their abilities when they feel supported by AI and that they are proficient in using it.
- *low*: Users feel less confident in their abilities when they over-rely on AI, compare themselves to the AI, or cannot successfully utilize the AI to reach their goals.
- ★ *self-blaming*: A failure of the AI is often framed by the user as their own failure to use it properly.

B.4.3 Trust in AI

Users depend on faith in AI's competence when they are unable to judge or predict its performance objectively.

B.5 Anthropomorphism

: Users try to reason about what AI is so that they know how to interact with it.

B.5.1 Objectivizing

Through a breadth of experiences with AI, programmers develop a more neutral view of its ontological status.

- *dispassion*: A programmer's lack of reactivity to interactions with AI interactions suggests an objective perspective.
- *resource*: Programmers with an objective view of AI tend to consider it as one of many tools rather than cure-all.

- *not blaming*: Programmers with an objective view of AI tend not to blame AI for challenges they face.
- *inconsistency*: Experiencing mixed success with AI over time promotes a balanced understanding of its nature.
- *teleology*: Programmers are more invested in the success of their programming efforts than the AI available to help them.

B.5.2 Humanizing

Users attribute human qualities to AI as a means to reason about its behavior.

B.6 Prior Knowledge

A user's background and experiences form the foundation for their interactions with AI.

B.6.1 Expectations

Lacking prior experience, users form predictions to anticipate AI interactions.

- *underestimating*: Users might predict that AI is less capable than it turns out to be.
- *overestimating*: Users might predict that AI is more capable than it turns out to be.

B.6.2 Biases

Users are influenced by others' ideas about AI interactions.

- *training*: How a user is trained to use AI affects how they will use it.

B.6.3 Competencies

Information a user has entering an AI interaction helps to guide their actions.

Appendix C

Qualitative Reporting Frequencies for Chapter 3

Tables C.1 and C.2 list the number of times each Cohort i participant and each prior qualitative study were quoted or paraphrased within the Results of Chapter 3. This helps to demonstrate that an intentional effort was made to balance the representation of different data sources.

Table C.1: Reporting frequency of direct quotes by participant × topic in Chapter 3

PID	Int. Patt.	Int. Qual.	Work Alloc.	Apprais.	Anthro.	Prior Know.	Total
i43	2	2	1	3	0	0	8
i9	3	1	2	1	1	0	8
i19	3	2	1	1	1	0	8
i47	3	0	1	2	1	1	8
i5	1	2	2	1	1	0	7
i24	1	2	1	3	0	0	7
i20	2	2	0	2	1	0	7
i11	2	3	0	1	0	1	7
i15	2	1	2	1	1	0	7
i42	3	1	2	1	0	0	7
i12	4	0	0	1	2	0	7
i4	4	0	1	1	1	0	7
i2	2	1	1	2	0	0	6
i10	2	1	3	0	0	0	6
i1	2	1	1	0	1	1	6
i13	1	1	0	1	1	1	5
i16	2	0	2	0	1	0	5

Table C.2: Reporting frequency of direct quotes by reference × topic in Chapter 3

Ref.	Int. Patt.	Int. Qual.	Work Alloc.	Apprais.	Anthro.	Prior Know.	Total
[11]	2	0	2	1	1	0	6
[10]	2	1	1	1	0	0	5
[4]	2	1	0	1	0	1	5
[3]	2	1	1	0	0	2	6
[9]	2	1	1	0	1	1	6
[1]	2	1	1	0	0	0	4
[2]	2	2	0	1	1	0	6
[8]	3	1	0	1	1	0	6
[6]	2	0	1	0	0	1	4
[5]	2	0	1	0	0	0	3
[7]	2	0	0	1	0	0	3

References

- [1] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: how programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7, 85–111, OOPSLA1. Number: OOPSLA1.
- [2] Gian Luca Scoccia. 2023. Exploring early adopters' perceptions of ChatGPT as a code generation tool, (Sept. 11, 2023). MAG ID: 4388214696. doi:10.1109/asew60602.2023.00016.
- [3] 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*, 1–15.
- [4] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J. Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–19.
- [5] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–12. isbn: 979-8-4007-1653-9. doi:10.1145/3631802.3631806.
- [6] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A large-scale survey on the usability of AI programming assistants: successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, (Feb. 6, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3608128.
- [7] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24: IEEE/ACM 46th International Conference on Software Engineering. ACM, Lisbon Portugal, (Apr. 12, 2024), 1–13. isbn: 979-8-4007-0217-4. doi:10.1145/3597503.3639187.
- [8] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. “it’s weird that it knows what i want”: usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31, 1, (Feb. 29, 2024), 1–31. doi:10.1145/3617367.

- [9] 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. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. IUI '23: 28th International Conference on Intelligent User Interfaces. ACM, Sydney NSW Australia, (Mar. 27, 2023), 491–514. isbn: 979-8-4007-0106-1. doi:10.1145/3581641.3584037.
- [10] 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 (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, (Apr. 28, 2022), 1–7. isbn: 978-1-4503-9156-6. doi:10.1145/3491101.3519665.
- [11] Justin D. Weisz et al. 2021. Perfection not required? human-AI partnerships in code translation. In MAG ID: 3154248444. (Apr. 14, 2021), 402–412. doi:10.1145/3397481.3450656.

Appendix D

Supplemental Material for Chapter 5

This appendix contains the supplemental Material for HBCU Student Perspectives on Identity, Persistence, and Code-Generating AI in CS Education: A Case Study.

D.1 Pre-study Questionnaire

- Select any racial or ethnic identification you wish to share. (White or Caucasian, Black or African American, Hispanic or Latinx/Latina/Latino, American Indian or Native American or Alaska Native, Asian, Native Hawaiian or Other Pacific Islander, Additional/Other, None of the above, Do not wish to identify)
- Select any gender identification you wish to share. (Non binary, Female, Male, Additional/Other, None of the above, Do not wish to identify)
- Select any additional gender or sexual identification you wish to share. (Gay or lesbian, Bisexual, Transgender, Queer or questioning, Additional/Other, None of the above, Do not wish to identify)
- The following question is completely optional. We hope that the technology we are studying can be assistive to students with disabilities and impairments to help them succeed in Computer Science, and this data may help to speed the adoption of this technology into CS

curricula. Select any impairments or disabilities with which you wish to identify. (Blindness or visual impairment, Deafness or hearing impairment, Mobility or dexterity or other physical impairment, Cognitive or attention impairment, Social or behavioral impairment, Other mental illness, Additional/Other, None of the above, Do not wish to identify)

- Please share anything else you are comfortable sharing regarding how you identify, such as aspects of your personality, culture, or family background.
- Please describe how you identify or position yourself within the field of computing. (i.e. what is your role, how do you think about yourself relative to others, what are your plans for the future relative to this field, etc.)
- Please select any role models you have in the field of computing: (Parent, Sibling, Extended Family, Acquaintance, Friend, Celebrity/Influencer, Other, None of the above, Do not wish to share)
- What factors do you attribute to your ability to persist in computing education, whether that be completing a course, or an entire degree?
- To the extent you are comfortable sharing, describe any benefits of having your identity in the context of computing.
- To the extent you are comfortable sharing, describe any barriers to your success in computing related to your identity.

D.2 Exit Interview Guide

- Member Checking: summarize their questionnaire responses and ask if their responses were understood correctly.
- Given what you experienced today, how do you think learning alongside a tool like GitHub Copilot would impact your success in computing education (either positively or negatively)?

- If others in school with you had access to this too, would that affect how or how much you would use it? How so?
- If you couldn't access this for free (but it was allowed) would you feel any pressure to pay for it to gain an advantage?
- Could having to pay for this be an issue for you if you felt a need to have it?
- Is there anything else you'd like to share about your identity or how the AI you worked with today relates to it?

Appendix E

Preprints for Chapters 4 and 7

At the time of archiving this dissertation, Chapters 4 and 7 were being peer-reviewed. Since they were closer to publication than other unpublished chapters, they were posted as preprints for the purposes of connecting this dissertation to them.

E.1 Preprint for Chapter 4

Chapter 4 entered third round review on April 20, 2026 with minor revisions expected to lead to later acceptance to *Human Factors: The Journal of the Human Factors and Ergonomics Society*. This version has been posted on arXiv (see [1]).

E.2 Preprint for Chapter 7

Chapter 7 received a conditional acceptance to the 22nd annual ACM Conference on International Computing Education Research (ICER 2026) on April 16, 2026 with required revisions expected to lead to later acceptance. The pre-revision version has been posted on arXiv (see [2]).

References

- [1] Nicholas Gardella, Matthew L. Bolton, and Sara L. Riggs. 2026. Relationships between trust, compliance, and performance for novice programmers using AI code generation. *_eprint: 2604.18948*. (2026).
doi:10.48550/arXiv.2604.18948.
- [2] Nicholas Gardella, James Prather, Juho Leinonen, Paul Denny, Raymond Pettit, and Sara L. Riggs. 2026. Fast and forgettable: a controlled study of novices' performance, learning, workload, and emotion in AI-assisted and human pair programming paradigms. *_eprint: 2604.18538*. (2026).
doi:10.48550/arXiv.2604.18538.

THE END



Thanks for reading!

- Nicholas & Carl