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

Nicholas Gardella
University of Virginia
Charlottesville, VA, USA
njg4ne@virginia.edu

Raymond Pettit
University of Virgina
Charlottesville, VA, USA
rp6zr@virginia.edu

Sara L. Riggs
University of Virginia
Charlottesville, VA, USA
sriggs@virginia.edu

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.

CCS CONCEPTS

• Social and professional topics ~ Professional topics ~ Computing education ~ Computing education programs ~ Computer science education ~ CS1 • Human-centered computing ~ Human computer interaction (HCI) ~ Empirical studies in HCI

KEYWORDS

Computer Science Education, Artificial Intelligence-driven Development Environment, Introductory Programming, CS1, AI Code Generators, GitHub Copilot, Novice Programmers, Generative AI

ACM Reference format:

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 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8-10,2024, Milan, Italy. ACM, New York, NY, USA, 7 pages.* https://doi.org/10.1145/3649217.3653615

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 [8,12]. 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 RELATED WORK

2.1 Human-AIDE Interaction

While many studies have evaluated AIDEs in isolation (e.g., [8,27,39]), 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 [30]. 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 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 [12]. 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 [32].

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.

2.3 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 frustration. 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 [33].

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.

2.4 Effects of Well-being in CS Education

Due to lack of skills and experience, programming tasks can be very challenging for novices [26]. Mismatches between students' abilities and the demands of a CS1 course can lead them to drop the course [3,31] 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 [25]. Programmers' emotions are often overlooked, but can have significant impacts on performance [11] and student experience [22]. Therefore, it is critical that educational programming experiences help to foster both positive emotions and strong self-efficacy in students.

3 METHOD

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.

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 3.3.1), a button would allow them to advance to the next task in the trial.

3.3 Tasks

3.3.1 Task Sourcing. Tasks were sourced from the 164-task, human-written HumanEval Dataset [9]. These tasks are a common tool for evaluating the coding performance of LLMs [27,28,39]. 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.

3.3.2 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.

3.3.3 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).

3.4 Experimental Design

3.4.1 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.

3.4.2 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.

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 1.

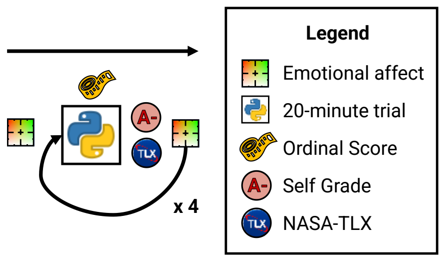

Figure 1: Experimental timeline for each trial that was repeated four times (x4).

3.6 Data Collection

3.6.1 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 [33] circumplex model of affect. Participants estimated their emotional valence (see Figure 2, top), followed by their emotional arousal (see Figure 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, bottom left), which was thoroughly explained to participants and visually aligned with the presentation of Likert choice bubbles.

3.6.2 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 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.

3.6.3 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.

3.6.4 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-.

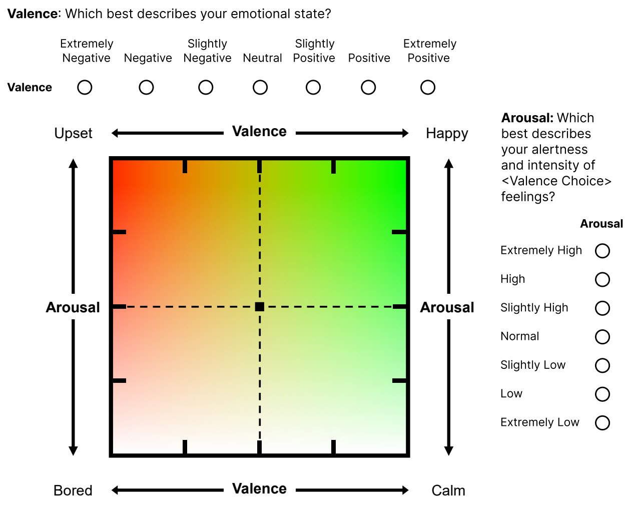

Figure 2: Exact questions asked to participants about their emotional affect alongside a complimentary visualization based on the Russell [33] circumplex of affect model.

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]) [23]. If any of these assumptions was not met, but the paired differences were symmetrical about some midline, the Wilcoxon signed rank test was used [23]. If asymmetry was clearly apparent, the sign test was used via a normal approximation of the binomial distribution [23,24]. Figure 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].

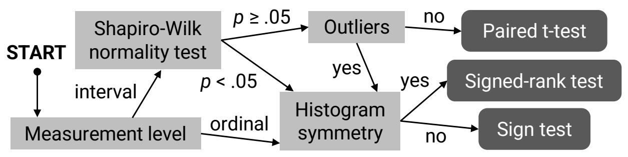

Figure 3: Decision tree for choosing statistical tests.

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 [10] 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 [29]. The significance threshold was set at α=.05 for all statistical tests.

4 RESULTS

4.1 Effects of AI Availability (RQ1)

4.1.1 Descriptive Statistics. Trends (Table 1) were toward improved performance with the AI, a more positive emotional valence effect, a more relaxing emotional arousal effect, decreases in most dimensions of NASA-TLX workload, and increases in self-grade (see Section 3.5.4 for letter to numerical grade conversion).

Table 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	$\bar{x} = -.088$	$\bar{x} = .118$	$\overline{\Delta x} = 0.206$
Δ Arousal	$\tilde{x} = 0$	$\tilde{x} = -.5$	$\widetilde{\Delta x} = -.5$
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	$\bar{x} = 7.647$	$\bar{x} = 7.912$	$\overline{\Delta x} = .265$

4.1.2 Statistical Tests. Tests were performed according to the prescribed method. The results can be found in Table 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: Statistical comparisons between solo and with AI trials for each measure. An asterisk (*) denotes significance.

	Statistic	p	d
Score	$\mathbf{z_{Wilcoxon} = 3.012}$	**.001 ∗**	**.368**
Δ Valence	$t(16) = .800$	.436	.098
Δ Arousal	$z_{Wilcoxon} = -1.329$	.191	−.162
TLX Mental	$\mathbf{t(16) = -2.487}$	**.024 ∗**	**−.304**
TLX Physical	$z_{Sign\ Test} = -.289$	.773	−.035
TLX Temporal	$t(16) = -1.338$	.200	−.163
TLX Performance	$t(16) = -1.248$	.230	−.152
TLX Effort	$\mathbf{z_{Wilcoxon} = -2.023}$	**.043 ∗**	**−.247**
TLX Frustration	$t(16) = -.563$	.581	−.069
Self-Grade	$t(16) = .765$	.455	.093

4.2 Effects of Experience (RQ2)

4.2.1 Descriptive Statistics. Trends (Table 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 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	$\tilde{x} = 2$	$\tilde{x} = 2$	$\widetilde{\Delta x} = 0$
	Self-Grade	$\tilde{x} = 8$	$\tilde{x} = 7.25$	$\widetilde{\Delta x} = 0$
with AI	Score	$\tilde{x} = 2$	$\tilde{x} = 3$	$\widetilde{\Delta x} = 1$
	Self-Grade	$\bar{x} = 7.118$	$\bar{x} = 8.706$	$\overline{\Delta x} = 1.588$

4.2.2 Statistical Tests. Tests were performed according to the prescribed method. The results can be found in
Table 4: Statistical comparisons between the first and second trial scores and self-grades for the solo and with AI conditions. An asterisk (*) denotes significance.

		Statistic	p	d
Solo	Score	$z_{Wilcoxon} = .090$	1.000	.016
	Self-Grade	$z_{Wilcoxon} = -.517$	.594	−.091
with AI	**Score**	$\mathbf{z_{Wilcoxon} = 2.029}$	**0.046 ∗**	**.348**
	Self-Grade	$\mathbf{t(16) = 2.567}$	**0.021 ∗**	**.440**

. 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 4: Statistical comparisons between the first and second trial scores and self-grades for the solo and with AI conditions. An asterisk (*) denotes significance.

		Statistic	p	d
Solo	Score	$z_{Wilcoxon} = .090$	1.000	.016
	Self-Grade	$z_{Wilcoxon} = -.517$	.594	−.091
with AI	**Score**	$\mathbf{z_{Wilcoxon} = 2.029}$	**0.046 ∗**	**.348**
	Self-Grade	$\mathbf{t(16) = 2.567}$	**0.021 ∗**	**.440**

5 DISCUSSION

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 more relaxing 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.

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].

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 3.3 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.

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 (1842490) & GRFP (1829004), 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*), January 05, 2023, New York, NY, USA. Association for Computing Machinery, New York, NY, USA, 1–5. . https://doi.org/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*, February 22, 2022, Providence RI USA. ACM, Providence RI USA, 668–674. . https://doi.org/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 Bull.* 41, 1 (March 2009), 153–157. https://doi.org/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*, August 12, 2013, San Diego San California USA. ACM, San Diego San California USA, 67–74. . https://doi.org/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*, 2023. 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 Bull.* 40, 1 (March 2008), 402–406. https://doi.org/10.1145/1352322.1352274

[7] Nigel Bosch and Sidney D'Mello. 2017. The Affective Experience of Novice Computer Programmers. *Int. J. Artif. Intell. Educ.* 27, 1 (March 2017), 181–206. https://doi.org/10.1007/s40593-015-0069-5

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, and Greg Brockman. 2021. Evaluating large language models trained on code. *ArXiv Prepr. ArXiv210703374* (2021).

[10] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences.* Academic press.

[11] 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)*, 2020. IEEE, 1–9. . Retrieved November 30, 2023 from https://ieeexplore.ieee.org/abstract/document/9274119/

[12] 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*), February 14, 2022, New York, NY, USA. Association for Computing Machinery, New York, NY, USA, 10–19. . https://doi.org/10.1145/3511861.3511863

[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? *Learn. Instr.* 16, 2 (2006), 104–121.

[14] Stefania Giannini. 2023. Reflections on generative AI and the future of education. 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.* Elsevier, 139–183.

[17] Toni Honicke and Jaclyn Broadbent. 2016. The influence of academic self-efficacy on academic performance: A systematic review. *Educ. Res. Rev.* 17, (2016), 63–84.

[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 31 3 Retrieved April* 6, (2015), 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. https://doi.org/10.1109/IEEESTD.2020.9084219

[20] Boris Iglewicz and David C. Hoaglin. 1993. *Volume 16: how to detect and handle outliers.* Quality Press.

[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*), April 19, 2023, New York, NY, USA. Association for Computing Machinery, New York, NY, USA, 1–23. . https://doi.org/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*, August 09, 2010, Aarhus Denmark. ACM, Aarhus Denmark, 77–86. . https://doi.org/10.1145/1839594.1839609

[23] Laerd Statistics. 2015. *Tutorials for SPSS Statistics.* Retrieved December 12, 2023 from https://statistics.laerd.com/

[24] Mike LePine. 2022. Chapter 9.4: Distribution Needed for Hypothesis Testing. In *College Statistics.* St. Clair College AA&T. Retrieved December 1, 2023 from https://ecampusontario.pressbooks.pub/sccstatistics/chapter/distribution-needed-for-hypothesis-testing/

[25] 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*, August 16, 2021, Virtual Event USA. ACM, Virtual Event USA, 252–265. . https://doi.org/10.1145/3446871.3469740

[26] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 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*), December 01, 2001, New York, NY, USA. Association for Computing Machinery, New York, NY, USA, 125–180. . https://doi.org/10.1145/572133.572137

[27] 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*), October 17, 2022, New York, NY, USA. Association for Computing Machinery, New York, NY, USA, 1–5. . https://doi.org/10.1145/3524842.3528470

[28] 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 Prepr. ArXiv220313474* (2022).

[29] Julie Pallant. 2007. *SPSS Survival Manual.* Open University Press New York, NY, USA.

[30] 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)*, 2022. IEEE, 754–768. .

[31] 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*, November 24, 2016, Koli Finland. ACM, Koli Finland, 71–80. . https://doi.org/10.1145/2999541.2999552

[32] Matei-Dan Popovici. 2023. ChatGPT in the Classroom. Exploring Its Potential and Limitations in a Functional Programming Course. *Int. J. Hum.-Comput. Interact.* (October 2023). https://doi.org/10.1080/10447318.2023.2269006

[33] James A. Russell. 1980. A circumplex model of affect. *J. Pers. Soc. Psychol.* 39, 6 (1980), 1161.

[34] John Sweller. 2011. Cognitive load theory. In *Psychology of learning and motivation.* Elsevier, 37–76. Retrieved December 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*), April 28, 2022, New York, NY, USA. Association for Computing Machinery, New York, NY, USA, 1–7. . https://doi.org/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. *Educ. Psychol.* 36, 2 (February 2016), 236–253. https://doi.org/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 Trans. Softw. Eng. Methodol.* 31, 2 (March 2022), 29:1-29:47. https://doi.org/10.1145/3487569

[38] Frank Yates. 1934. Contingency tables involving small numbers and the χ 2 test. *Suppl. J. R. Stat. Soc.* 1, 2 (1934), 217–235.

[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*, 2022. 62–71. .