# Is AI the better programming partner? Human-Human Pair Programming vs. Human-AI pAIr Programming

Qianou Ma*
Carnegie Mellon University
Pittsburgh, USA
qianouma@cmu.edu

Tongshuang Wu
Carnegie Mellon University
Pittsburgh, USA
sherryw@cs.cmu.edu

Kenneth Koedinger
Carnegie Mellon University
Pittsburgh, USA
koedinger@cmu.edu



## ABSTRACT

The emergence of large-language models (LLMs) that excel at code generation and commercial products such as GitHub's Copilot has sparked interest in human-AI pair programming (referred to as "pAIr programming") where an AI system collaborates with a human programmer. While traditional pair programming between humans has been extensively studied, it remains uncertain whether its findings can be applied to human-AI pair programming. We compare human-human and human-AI pair programming, exploring their similarities and differences in interaction, measures, benefits, and challenges. We find that the effectiveness of both approaches is mixed in the literature (though the measures used for pAIr programming are not as comprehensive). We summarize moderating factors on the success of human-human pair programming, which provides opportunities for pAIr programming research. For example, mismatched expertise makes pair programming less productive, therefore well-designed AI programming assistants may adapt to differences in expertise levels.

## KEYWORDS

Pair Programming, LLM, Human-AI Interaction, Copilot, AI-Assisted Programming

## 1 INTRODUCTION

Pair programming was first introduced in the 1990s as part of the Agile software development practice [9]. In its original definition, pair programming describes the practice of two programmers working together on the same task using a single computer, keyboard, and mouse. One programmer in the pair, the "driver," performs the coding (typing) and implements the task, while the other programmer, the "navigator," aids in planning, reviewing, debugging, and suggesting improvements and alternatives. Over time, pair programming has evolved and adapted to different contexts and purposes. Now, it is used in a wide range of settings, including education, industry, and open-source software development [5, 83].

*Corresponding author

Recent advances in code-generating large-language models (LLMs) have led to the widespread popularity of commercial AI-powered programming assistance tools such as GitHub Copilot [26], which advertises itself as "your AI pair programmer." For pAIr programming, instead of two humans working on a single computer, it is the programmer and the LLM-based AI that work together on the same task. The shift in the paradigm raises the questions: *Is the AI programming partner comparable to a human pair programmer? Are they applicable to the same contexts, can they achieve similar or better performance, and should people interact with them in the same way?*

In this work, we delve into the current state of research on human-human and human-AI pair programming to uncover their similarities and differences, and we hope to inspire better evaluations and designs of code-generating LLMs as a pAIr programmer. We start by reviewing the application context, methods, and tasks for both human-human and human-AI pair programming literature (Section 2), then dive into fine-grained comparisons of their measurements of success (Section 3), as well as the contributing moderators, e.g., pair compatibility factors like expertise (Section 4).

We find that (1) prior work on both pair programming paradigms has observed mixed results in *quality*, *productivity*, *satisfaction*, *learning*, and *cost*, (2) pAIr programming has yet to develop comprehensive measurements, and (3) key factors to pAIr's success have been largely unexplored.

Building on our exploration, we then discuss views and challenges of characterizing AI as a pair programmer, and elaborate on future opportunities for developing best practices and guidelines for human-AI pair programming (Section 5). First, we argue that moderating factors that bring challenges to human-human pair programming (e.g., compatibility and communication) unveil opportunities to improve human-AI pair programming. It can be promising to exploit the differences between a human and an AI partner (e.g., more customizable expertise level and more adaptable communication styles) to design for more successful human-AI pair programming experiences. Second, we encourage future research to explore the best deployment environment for

human-AI pair programming. While most human-AI pair programming works have focused on assisting professional developers, we hope to inspire more future works in the learning context (or, student-AI pair programming), and we highlight potential challenges involved.

## 2 CONTEXTS, METHODS, AND TASKS

Human-human pair programming originated as a practice in the software engineering industry [9] and then become a popular collaborative learning practice in classrooms [83]. Therefore, in this paper, we compare human-human and human-AI pair programming in both the industry and education contexts, as they are the most common contexts.

We adhere to the original definition of human-human pair programming to closely resemble human-AI interaction on a single device. Other modes of interaction exist for comparing human and human-AI teams in programming tasks, such as computer-mediated collaborative learning [71] and distributed pair programming [19], but they are beyond the scope of this paper.

For human-AI pair programming, most current works have been evaluating Copilot using case studies (e.g., [12]) or experimental studies (e.g., [84]) with experienced programmers in the industry. Similar to human-human pair programming, researchers tried to mimic a realistic professional development environment in their task setup. For example, Barke et al. [8] invited 20 participants, mostly doctoral students and software engineers, to complete tasks such as developing Chat Client and Server. However, there is a lack of non-invasive field observation studies like what human-human pair programming studies have done [65, 75].

Few recent works have explored using LLM-based programming environments or Copilot with students. For example, Kazemitabaar et al. [39] used a controlled experimental study with 69 novice students from 10 to 17 years old working on 45 Python code-authoring and code-modifying tasks. However, existing works on human-AI pair programming are mostly in lab experiments, and there is still a lack of large-scale study [51] and classroom deployment [57, 87] as in the human-human pair programming literature.

When setting up comparison groups, existing pAIr programming works have been comparing the human-AI pair against human-human [35] or human solo (e.g., compare developers' work when they use Copilot or the default code completion tool) [84]. No current study sets up a three-way comparison for human-AI, human-human, and human-solo.

*Summary:* In comparison to human-human pair programming works, existing pAIr studies lack realistic deployment in the workspace or classroom, and a larger sample size would also be desirable. Researchers of both pair programming paradigms use various study designs to examine what affects the effectiveness of pair programming. In Section 3 and Section 4, we compare the variables and measurements they used to further uncover what is lacking in pAIr studies.

## 3 MIXED OUTCOMES

Literature reviews of human-human pair programming have suggested various benefits as well as mixed effects. In the industry context, according to Alves De Lima Salge and Berente [5], pair programming improves code quality, increases productivity, and enhances learning outcomes. However, according to Hannay et al. [31], pair programming improves quality and shortens duration, but it increases effort, higher quality comes at the expense of considerably greater effort, and reduced completion time comes with lower quality. In the education context, pair programming brings benefits including higher quality software, student confidence in solutions, increased assignment grades, exam scores, success/passing rates in introductory courses, and retention [29, 52, 83]. All the reviews on human-human pair programming acknowledged that even though meta-analysis can show an overall trend and significant effect size, individual studies could report contradictory outcomes (see examples in Table 1).

For human-AI pair programming, existing works mainly focus on quality, productivity, and satisfaction, and already demonstrated mixed results in quality and productivity [8, 35, 84] (see examples in Table 1). Additionally, there is not enough research for a comprehensive review, so we cannot reach any conclusion on the effectiveness of human-AI pair programming yet. It is also hard to compare the human-human and human-AI pair programming literature, as they differ in what outcomes and measurements they adopt.

Therefore, in the top rows of Table 1, we listed the most common outcome variables in both literature (*quality*, *productivity*, *satisfaction*, *learning*, and *cost*) and some sample work to demonstrate the mixed outcomes and various measures. We elaborate on the variety of ways to measure some of the listed outcomes as follows.

### 3.1 Quality

In human-human pair programming literature, quality can be measured using defect density, perceptual effort measure, readability, functionability, the number of test cases passed, code complexity, scores, expert opinions, etc. [5, 70, 79].

### 3.2 Productivity

In human-human pair programming literature, duration, effort, and productivity are all types of "efficiency" outcomes that involve time and accomplishment. Productivity can be measured in terms of the number of completed tasks in a fixed unit of time, duration can be measured as the amount of elapsed or total time used to complete a fixed number of

**Table 1: Comparison of Outcome Variables and Moderators for Human-Human Pair Programming vs. Human-AI pAIr Programming**

| Outcomes | Human-Human vs. Human Solo | Human-AI (Copilot) |
|---|---|---|
| Quality | 😊 significantly lower defect density for complex code<br>😐 no difference for simpler code [76]<br>😊 significantly higher percentage of test cases passed [86] | ☹ vs. Human-Human: more lines of code deleted in next session (lower quality) [35]<br>😊 vs. Human Solo: significantly improve correctness score and reduce encountered errors for novice students [39]<br>😐 vs. Human Solo: no significant difference in task success [63] or task success rate in given time [84] |
| Productivity | ☹ significantly fewer lines of code per person hour writing simpler code,<br>😐 no significant difference writing more complex code [76]<br>😊 29% shorter time to complete task (pair speed advantage = 1.4) [60] | 😊 vs. Human-Human: more lines of added code [35]<br>😊 vs. Human Solo: 55.8% reduction in completion time [63]<br>😊 vs. Human Solo: significantly increase task completion and reduce task completion time for novice students [39]<br>😐 vs. Human Solo: no significant difference in the task completion rate in given time [84] |
| Satisfaction | 😊 higher self-ratings of satisfaction [70]<br>☹ students with greater self-confidence and self-efficacy less enjoy the pair programming experience [81] | 😊 vs. Human Solo: higher self-ratings of satisfaction [12, 37, 84] |
| Learning | 😊 higher grades, exam scores [57], and retention [52]<br>😊 significantly higher gains in exam performance in female students than male students [47] | 😐 vs. Human Solo: no significant difference in immediate and retention post-test performance of novices, students with more prior experiences have more learning gains from AI code generator [39] |
| Cost | ☹ increased management workload to match, schedule a pair, resolve collaboration conflict, assess individual contributions, etc. [4]<br>😊 reduced teaching staff workload (grading one assignment from a pair) [86] | No experiment yet. Bird et al. [12], Vaithilingam et al. [84] hypothesized that human-AI may lead to more unnecessary debugging vs. Human Solo |

| Moderators | Human-Human vs. Human Solo | Human-AI (Copilot) |
|---|---|---|
| Task Types & Complexity | Complex task improve quality, simple one does not [76]; debugging is perceived as less enjoyable or effective than comprehension or refactoring [16] | - |
| Compatibility (E.g., Expertise) | Random pairing led to incompatible partners and conflicts during work [57]. Expertise: improve quality more effectively if pair is similarly skilled [70]; less-skilled students learn more and enjoy more [16, 47]; if knowledge gap is large, less-skilled programmers may tend to be more passive and disengaged [17] | - |
| Communication | Conversations with intermediate-level details contribute to pair programming success [24]; different types of discourse lead to more attempts or more debug success [55] | - |
| Collaboration | Over-reliance leads to conflicts and impedes satisfaction and learning, as work is entirely burdened on one partner [57, 87]; educators recommend regular role-switching to ensure equitable learning in collaboration [83] | - |
| Logistics | Scheduling difficulties [11], teaching & evaluating individual responsibility and accountability are important to collaboration success [67], but can lead to increased management costs [4, 79] | - |

tasks to a certain standard, and effort can be measured as twice the duration, the person-hours required, etc. [5]. We use productivity as an aggregated outcome variable of different measures, for consistency with the human-AI literature.

In current human-AI works, some measures are arguably too simplified as evaluation metrics, for example, Imai [35] used the number of lines of added code as the measure of productivity; however, the nature of interaction with Copilot (tab to accept suggestions) is likely to contribute to more added lines in the human-Copilot condition, and how valid would it represent the notion of productivity is questionable.

Note that some researchers have examined programmers' perceived productivity when working Copilot and found that it most strongly correlated with the general acceptance rate of AI-generated code [90]. This is not included in Table 1 to stay consistent with the human-human pair programming literature, as perceived productivity is a different measure than actual productivity.

## 3.3 Learning

In human-human pair programming literature, learning can be assessed by quantitative measures such as assignment grades, exam scores, passing rate, and retention rate, or qualitative measures of higher-order thinking skills [29, 52, 83].

## 3.4 Cost

In terms of cost, there is the observation that participants faced challenges in understanding and debugging Copilot's generated code, which leads to the hypothesis that human-AI pair programming could cost additional efforts and hinder programmers' task-solving effectiveness [12, 84]. However, Dakhel et al. [21] shows that although Copilot's code could be less correct than human code, its bugs are easier to debug than human errors. There is currently no work that experimentally characterizes the costs of human-AI pair programming.

*Summary:* The literature on human-human pair programming has shown mixed results in many outcome variables, including *quality*, *productivity*, *satisfaction*, *learning*, and *cost*. For human-AI pair programming, or mostly human-Copilot in this paper, there are still only few works with incomprehensive measures, but a mixed outcome is also observed. We further review the potential causes of mixed outcomes of both modes of pair programming in Section 4.

# 4 MODERATORS

In search of the explanations of the cost-benefit of human-human pair programming experiences, researchers have found moderators such as *task type & complexity* [31], *compatibility* factors like expertise [6, 67], *communication* [17, 24, 65], *collaboration* factors like over-reliance and role-switching [30, 70, 87], and *logistics* difficulties including scheduling and training [11, 31] (as shown in the bottom rows of Table 1).

These key factors influence the success of human-human pair programming. If they work well, pair programming helps programmers catch errors more easily, solve problems more quickly, review code more thoroughly, and produce overall higher-quality code; it also promotes knowledge sharing among team members, which can lead to a more cohesive and effective team. If not, challenges such as scheduling and finding suitable pairs with compatible working styles usually result in a low cost-efficiency in pair programming, and slow down the development process if there are conflicts or disagreements between pair partners [11, 18].

For human-AI pair programming's moderators, much was unexplored – we do not know what could make human-AI pair programming more or less effective. Therefore, in this section, we discuss the key moderators that are examined in the human-human pair programming literature, and individual examples of moderating effects are provided in Table 1.

## 4.1 Task Types & Complexity

For task type and task complexity, Chaparro et al. [16] found that debugging tasks lead to less satisfaction and perceived efficacy compared to comprehension and refactoring tasks. Hannay et al. [31] found that the duration is shorter for low complexity tasks, at the expense of lower quality results, and quality is higher when complexity is higher, but it requires considerably greater effort. Arisholm et al. [6] found that the moderating effect of complexity also depends on the expertise of the pair, where "benefits of correctness on complex system apply mainly to juniors, whereas the reductions in duration to perform the tasks correctly on the simple system apply mainly to intermediates and seniors."

## 4.2 Compatibility

Salleh et al. [70] listed multiple factors for pair compatibility, such as personality, perceived skills, actual skills (expertise), self-esteem, gender, and work ethic. Thomas et al. [81] found that paired students with similar self-confidence levels produce their best work. Hannay et al. [30] found that Big Five personality traits only have modest predictive value on pair programming performance, and expertise, task complexity, and country have stronger prediction power in comparison. There also seems to be evidence that women benefit from pair programming more than men do [29, 67].

Expertise as a compatibility factor has been extensively studied in the human-human pair programming literature. For example, researchers found that a student pair performs the best when their expertise is similar [70] and students preferred to be paired with similarly skilled partners [16]. However, in industry, Jensen [36] reported that when both members were near the same capability level and strongly opinionated, the collaboration was counter-productive and troublesome.

In the introductory programming context, Lui and Chan [45] found that pairing up novices results in a larger improvement in productivity than pairing up experts. However, there are concerns about the risk of "the blind leading the blind" if they don't have an expert to consult with [4]. Researchers also found that less-skilled students learn and enjoy more than more-skilled students in pair programming [16, 47]. However, when the knowledge gap is too large, students can be less satisfied and the benefits of quality may be smaller [60]. Chong and Hurlbutt [17] reported that a novice programmer collaborating with an expert may become disengaged, have lower self-esteem, and be afraid of slowing down or annoying their more-skilled partner [4].

## 4.3 Communication

According to Freudenberg et al. [24], "the key to the success of pair programming [is] the proliferation of talk at an intermediate level of detail in pair programmers' conversations." Researchers also found that pair programming eliminates distracting activity and enables programmers to focus on productive activity [75], which could be why engaging communications contribute to the success of pair programming. Murphy et al. [55] used transactive analysis to break down communication by different types of transactions, and they found that attempting more problems associated with more completion transactions and debugging success correlated with more critique transactions. Some other works pointed out the social support aspect of communication [17] and an explanation effect where the verbalization of the thought process makes it clearer [12].

In human-human pair programming, programmers spend about 1/3 of the time primarily focusing on communication [65], which forces them to concentrate, rationalize, and explain their thoughts [31, 75]. In human-AI pair programming, Mozannar et al. [53] has shown that an analogous 1/3 amount of time is spent communicating with Copilot, such as thinking and verifying (22.4%) Copilot's suggestion, which may be replicating the self-explanation effects in some ways, and prompt crafting, which takes 11.56% of the time. These activities are arguably efforts to understand and communicate with Copilot. However, there is no other human to co-verify the answers, and there is no study that evaluate the communicative nature of human-Copilot interaction as human-human pair programming.

### 4.4 Collaboration

How well partners collaborate have been important factors that affect pair programming effectiveness [4, 79], and cooperative behavior and positive interdependence are key to pair programming success [67].

Collaboration can fail in various ways in a human-human pair. For example, the free-rider problem, where the entire workload is on one partner while the other remains a marginal player, can result in less satisfaction and learning [57, 87]. In human-AI pair programming, educators are worried that easily available code-generation tools may lead to cheating, and over-reliance on AI may hinder students learning [10]. However, no study has formally evaluated it.

For human-human pair programming, there is a suggested collaboration pattern of role-switching – two software developers periodically and regularly switch between writing code (driver) and suggesting code (navigator), aiming to ensure that both are engaged in the task and alleviate the physical and cognitive load borne by the driver [5, 65].

Some researchers Freudenberg et al. [24] argue that the success of pair programming should be attributed to communication rather than "the differences in behavior or focus between the driver and navigator," as they found both driver and navigator worked on similar levels of abstraction. Nevertheless, instructors still recommend drivers and navigators to regularly alternate roles to ensure equitable learning experiences [83].

In human-AI interaction, given Copilot's amazing capability to write code in different languages, some have argued that Copilot can take on the role of the "driver" in pair programming, allowing a solo programmer to take on the role of the "navigator" and focus on understanding the code at a higher level [35]. However, while it is possible for humans to offload some API lookup and syntax details to Copilot, humans still need to jump back into the driver's seat frequently and fluidly switch between the thinking and writing activities [53]. It is ultimately the human programmer's sole responsibility to understand code at the statement level [72].

### 4.5 Logistics

Logistical challenges, including scheduling difficulties, teaching and evaluating collaboration for the pair, and figuring out individual accountability and responsibility [11, 67], can add to the management cost of human-human pair programming [4, 79].

In human-AI pair programming, some may argue that the human is solely responsible in the human-AI pair [72], but the accountability of these LLM-based generative AI is still under debate [10]. There may be new logistics issues for the human-AI pair, such as teaching humans how to best collaborate with Copilot. There could also be unique challenges as in every human-AI interaction scenario, such as bias, trust, and technical limitations – much to be explored. More study would be needed to empirically and experimentally verify the moderating effects of different variables in human-AI pair programming.

*Summary:* Human-human pair programming literature have found moderators including *task type & complexity*, *compatibility*, *communication*, *collaboration*, and *logistics*. However, there is a lack of in-depth examination of potential moderating effects in current pAIr works.

## 5 DISCUSSION AND FUTURE WORK

### 5.1 LLM, Your pAIr Programmer?

Before the occurrence of LLM-based tools that claim to be "your AI pair programmer [26]," people have already been developing AI-powered systems to assist programmers, such as code completion tools (e.g., Tabnine), code refactoring and formal verification systems, and code synthesis and debugging tools. The evaluation focus has mostly been on usability design, cost-efficiency, and productivity [53, 56], but not on the feasibility of using these AI-assisted programming tools as the pair programming partner.

With recent advancements in generative LLM technologies, commercial AI tools like Copilot which are capable of offering real-time code suggestions and feedback beyond auto-completion seem to have a closer resemblance to a pair programming partner [12]. Many studies have evaluated and critiqued Copilot's ability to generate correct, efficient [21, 58], secure [7, 62], readable [3], and verifiable [88] code. Without doubt, Copilot generates defects and errors in its suggested code, but humans are far from error-free either. A programmer cannot be and does not need to be perfect to bring benefit into the pair programming experiences, but would Copilot be qualified as a programming partner?

In answering this question, researchers start to look into the interaction dynamics between programmers and the claimed AI pair programmer. Some researchers argue against the characterization of AI-assisted programming as pair programming. They believe that the analogy to human-AI pair programming is rather superficial, as what makes human-human pair programming effective (e.g., productive communication) disappear in human-AI pair programming. According to Sarkar et al. [72], "LLM-assisted programming ought to be viewed as a new way of programming with its own distinct properties and challenges."

We used the phrase "human-AI pAIr programming" in this paper, simply because we adopt the definition of pair programming that a pair work on the same device and the same task, so we can conveniently compare human and AI as a pair programming partner. As reviewed in Section 3 and Section 4, Copilot and a human partner share a lot of similar outcomes in pair programming, but the moderators for human-AI pair programming are less examined. We believe that this comparison is meaningful in that it helps us derive insights to keep improving LLM-based programming tools.

Note that in this paper, we mostly covered studies using the VSCode Extension Copilot. Tools like ChatGPT may support the communication aspect better than Copilot [82], and there are also Bard developed by Google [27] and an experimental version of Copilot Labs by Github [25], which support more functionalities such as fix bug, clean, and customizable prompts. Those tools may already improve the human-AI pair programming interaction in some ways, so future studies could also compare across a variety of LLM-based programming tools.

There is another challenge to describing AI as a pair programmer, following the debate on anthropomorphizing user interfaces [74] and ongoing discussion as AI demonstrates increasing capabilities to replicate human behaviors [43, 80]. The concern is that anthropomorphized AI could mislead designers and deceive users, impede user agency and responsibility, have deeper ethical and social risks, and may not be more effective anyways.

However, in educational literature, researchers have been trying to make agents provide naturalistic and human-like interactions with students, using teachable agents [13, 59], pedagogical agents [44, 46, 49], conversational agent [69, 71], etc. Kuttal et al. [41] explored the trade-off of using a human vs. AI agent as the pair programming partner. They found that human-human and human-AI led to similar productivity, code quality, and self-efficacy results, and students "trusted and showed humility towards agents." They also found that AI agents successfully facilitated knowledge transfer while failing at providing logical explanations or discussions.

Those anthropomorphized agents mostly seem to be effective in improving learning and motivation [32, 73]. Some explained the effects using social agency theory [49], cognitive load theory [44], and social cues related multimedia learning principles [48]. How well can we apply these theories to LLM-supported AI agents, and what's different in industry versus educational context would be interesting to explore. More works are welcomed to create a shared vocabulary for this field.

## 5.2 LLM, A Better pAIr Programmer?

As reviewed in Section 3, previous literature has explored a variety of measures to evaluate different aspects of human-human pair programming, while the current exploration in human-AI pair programming is quite limited. Murillo and D'Angelo [54] have proposed evaluation metrics for LLM-based creative code writing assistants in software engineering. More works could use more valid measures in the human-human pair programming literature to explore how to best help humans and LLM-based AI programming assistant collaborate together. It would also be interesting to have a study setup with three conditions – human-human, human-AI, and human solo – working on the same task.

Previous literature suggested some key factors in the success of human-human pair programming, as summarized in Table 1. These moderators that cause challenges for human-human pair programming may yield opportunities to explore in human-AI pair programming (Table 2). For example, self-efficacy can lead to a difference in satisfaction [81] and gender can lead to a difference in learning [47], do these compatibility moderators influence pAIr too? Can we improve pAIr outcomes using insights derived from human-human literature (e.g., simulate an AI partner with similar self-efficacy levels and the same gender)? Therefore, in general, we can ask the following questions for future works: *Could these factors be implemented for human-AI pair programming? Would they make human-AI pair programming more effective, less effective, or have no influence, and why?*

*Task Types & Complexity.* As we know from the human-human pair programming literature, a good collaborative task of the right complexity is important, but creating or choosing such tasks can be difficult. Meanwhile, LLMs help educators efficiently generate instructional materials such as questions [85], question-answers [40], feedback [20], and hints [61], which could be of similar quality as human-authored content. There is also work that suggested the preliminary success in using LLM to break down problems into subquestions [78]. Therefore, based on the insight from human-human pair programming literature and the known capacities of LLM, there is an open question to explore in human-AI pAIr programming: can LLM be configured to generate a task type with collaborative learning goals and customize task complexity for a programmer?

**Table 2: Challenges in Human-Human Pair Programming *Yield* Opportunities for Human-AI pAIr Programming**

| Moderating Factors | Human-Human Challenges | Human-AI Opportunities |
|---|---|---|
| **Task Types & Complexity**: pair work better if the task is not too simple and good for collaboration [16, 76] | Hard to design suitable tasks of appropriate complexity level | AI may be used to generate collaboration tasks and adjust tasks complexity |
| **Compatibility**: pairs with similar skill levels and compatible working styles work better [16, 70] | Hard to find a similarly skilled or compatible partner | AI partner should adjust to human skill level and adapt to be compatible with different people |
| **Communication**: pairs work better with productive conversations [24], and critiques lead to more debugging success [55] | Hard to teach effective communication and constructive criticism | AI partner should support productive conversations and provide critiques |
| **Collaboration**: pairs work better with positive interdependence [67] and clear and balanced responsibilities [57] | Hard to teach collaboration and prevent free riders | AI should support positive social interactions and collaboration and avoid over-assist that eliminates human's need to engage |
| **Logistics**: pair programming is costly to implement because of management challenges [4, 79] | Hard to schedule and assess individual contributions in a pair | Scheduling is no longer a problem, but humans should be accountable and responsible when using AI-generated code |

*Compatibility - Expertise.* In terms of the compatibility factor expertise, the pair programming literature suggests that matching partners with a similar level of expertise may be the best in promoting productivity and learning [5, 16, 31]. Evaluation studies show that GPT3-based models can be an above-average student in a CS1 classroom [22, 68] and its performance gets worse when the code becomes more complicated [89]. GPT4 even does better at solving introductory and basic programming problems (although its correctness is still not comparable to a developer in practice) [14]. We can also purposefully generate bugs and let the models make mistakes [38], so potentially, we may create an AI partner with a similar skill level to novice students. Future works can examine how to configure AI to adapt to student's skill levels and whether it will be effective or not.

*Other Compatibility Factors.* Researchers have explored how to let LLMs generate interaction based on a designed persona and reasonably replicate human behavior [1, 34], and in education, Cao [15] let LLMs interact with students while role-playing as different fictional characters to help reduce students' anxiety and increase motivation. There are possibilities to personalize an AI partner with different personality traits or the other pair compatibility factors like gender, ethnicity, and self-esteem that Salleh et al. [70] proposed. Potentially, it can be used to increase programmers' motivation and/or engagement, but how useful it is for human-AI pair programming is yet to be examined.

*Communication.* For communication, we know the social aspect of a conversation matter [17] and that some types of discourse could be more effective to facilitate debugging [55] in human-human pair programming. Therefore, since LLM-based tools such as ChatGPT are able to simulate social interaction, it would be interesting to explore if LLM can support different types of communication, can the different components of communication be replicated in an LLM-based programming assistant, and whether it is effective or not.

*Collaboration.* In terms of collaboration, it is frequently reported that creating smooth collaboration is challenging in both industry [11] and educational context [57, 87]. Given that the free-rider problem reduce pair programming's effectiveness [57] and regular role-switching potentially alleviates the driver's cognitive load and ensures balanced learning outcomes [5, 83], it would be interesting to explore if LLM-based AI can be configured to avoid over-help, support role-switching, and how to best support the human-AI pair to collaborate.

*Logistics.* Logistics-wise, the use of Copilot as a programming partner may have the special advantage of avoiding scheduling logistics, but there are also concerns of accountability that need to be addressed [12, 22]. In general, there will be ethical risks and social implications of using AI in pair programming at the workplace and in educational contexts, which needs deeper examination in future works.

## 5.3 LLM, Students' pAIr Programmer?

As reviewed in Section 2, most current studies that evaluate the efficacy of Copilot are conducted with experienced software developers. If we estimate Copilot's problem-solving abilities as an average student in introductory programming classes, evaluating its performance when pairing up with a professional software developer with much more expertise may not bring enough benefit to the professional. Therefore, working with LLM's current capabilities, it seems like a student-AI pair programming setup would be the most

promising to explore, so the next question is: how should we best support student-AI pair programming?

*Re-prioritize programming skills.* Co-working with AI requires a special skill set, and future work could explore how to support students to better develop these crucial skills. Bird et al. [12] argued that the popularity of LLM-based programming assistants will result in the growing importance of reviewing code as a skill for developers. Nonetheless, in Perscheid et al. [64]'s interview, none of the professional developers remembered training on debugging at school. There is already rich literature on debugging and testing instructions [2, 50, 77], but logistical challenges like the lack of instructional time still exist [23, 50], and educators need to better prepare students with debugging and testing skills needed to work with unreliable AI.

*Integrate AIEd frameworks.* On the theoretical side, Holstein et al. [33] developed a framework to map ways to mutually augment humans and AI in education, for example, by augmenting interpretation, action, scalability, and capacity. Future works can use existing theories in the AI education space to improve the design of the AI pAIr programming partner, and further investigate if LLMs bring new focus and affordances to previous human-AI education frameworks.

*Support explanation and communication with students.* Previous attempts of using AI agent as pair programming partner have shown some preliminary success in knowledge transfer and retention [28, 69], and the limitation discussed was the lack of discussion and explanation [41]. Nowadays, as an LLM-based agent can support more natural interaction and provide good quality explanations in the introductory programming context [42], it would be interesting to explore if LLM-based AI could resolve some limitations mentioned in pedagogical and conversational agent works before. Self-reflection and explanation techniques may also be adopted to make up for the communication aspect as in human-human pair programming.

*Match expertise with students.* As discussed in Section 4, matching expertise is a tricky problem. Lui and Chan [45] found that expert-expert pair may not gain as much of an advantage over an expert solo programmer, in comparison to novice-novice pair vs. a solo novice. Meanwhile, pairing two novices together raise concerns of "the blind leading the blind," but pairing a novice with an expert may lead to lower self-esteem of the novice [4]. Given all these complexities, when it comes to a student-AI pair and when we only care about the student's learning gains, there are a lot of research questions to ask. If we have full control of the perceived skill level of the AI partner, should we configure it to be similar to the student, slightly higher-skilled, or a lot better? Would it be beneficial to have both a peer AI agent but also a tutor AI agent to assist if students get stuck?

*Avoid over-helping students.* For programming learners, it would be important to configure the LLM-based programming assistant to avoid over-help. In the few studies that examined novice interaction with Copilot [66] or a customized programming environment based on LLM-based code generation model Codex [39]. Prather et al. [66] found that novices do have unique interaction patterns with Copilot and a tendency to rely on and trust the generated code too much. Kazemitabaar et al. [39] discussed design implications including control over-use and support complete novices. There have also been concerns about academic integrity and changing perception of learning when LLM-based programming tools become easily accessible to students [10, 66, 68], which need further explorations for student-AI pair programming.

*Boost students' self-confidence.* Last but not least, pair programming has been shown to benefit students with lower self-efficacy and self-confidence levels [81] and women [47] more, which could make it a pedagogical tool to engage more vulnerable or underrepresented populations in CS. When an AI is introduced in pair programming, would the same benefit retains? How should we present the AI differently to make it compatible to students with different confidence levels? How do we mitigate the risks of unreliable but seemingly authoritative AI? LLMs may be an opportunity to address some existing challenges in student-student pair programming (as summarized in Table 2), but there are still a lot of open questions to ask.

## 6 CONCLUSION

This paper has discussed the concept of human-AI pair programming (pAIr programming). We found that both human-human and human-AI pair programming have benefits and challenges, but current research did not give us a clear answer on the efficacy of human-AI pair programming. Human-human pair programming literature yield insights on what study designs should pAIr researchers adopt (e.g., more realistic observations), what outcomes and measures should pAIr researchers use to evaluate their work (e.g., use more valid quality and productivity measurements, and further investigate cost), and what moderators should pAIr researchers consider to further analyze the pAIr process and improve pAIr design (e.g, compatibility, communication, etc.).

In conclusion, more valid and comprehensive measurements are needed to evaluate pAIr, more comparisons can be drawn between human-human vs. human-AI pair programming, and more works can explore how to best support LLM-assisted programming with insights from the rich literature on human-human pair programming.

## ACKNOWLEDGMENTS

Thanks to Ken's lab members for giving feedback on this work. Thanks to Dr. Stephen MacNeil for coming up with the creative "pAIr" keyword for this project.

## REFERENCES

[1] Gati Aher, Rosa I Arriaga, and Adam Tauman Kalai. 2022. Using Large Language Models to Simulate Multiple Humans and Replicate Human Subject Studies. (Aug. 2022). arXiv:2208.10264 [cs.CL] http://arxiv.org/abs/2208.10264

[2] Wolfgang Ahrendt, Richard Bubel, and Reiner Hähnle. 2009. Integrated and Tool-Supported Teaching of Testing, Debugging, and Verification. In *Teaching Formal Methods*. Springer Berlin Heidelberg, 125–143. https://doi.org/10.1007/978-3-642-04912-5_9

[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*. Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/3551349.3560438

[4] Mustafa Ally, Fiona Darroch, and Mark Toleman. 2005. A framework for understanding the factors influencing pair programming success. In *Extreme Programming and Agile Processes in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 82–91. https://doi.org/10.1007/11499053_10

[5] Carolina Alves De Lima Salge and Nicholas Berente. 2016. Pair Programming vs. Solo Programming: What Do We Know After 15 Years of Research?. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*. 5398–5406. https://doi.org/10.1109/HICSS.2016.667

[6] Erik Arisholm, Hans Gallis, Tore Dyba, and Dag I K Sjoberg. 2007. Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Trans. Software Eng.* 33, 2 (Feb. 2007), 65–86. https://doi.org/10.1109/TSE.2007.17

[7] Owura Asare, Meiyappan Nagappan, and N Asokan. 2022. Is GitHub's Copilot as Bad as Humans at Introducing Vulnerabilities in Code? (April 2022). arXiv:2204.04741 [cs.SE] http://arxiv.org/abs/2204.04741

[8] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2022. Grounded Copilot: How Programmers Interact with Code-Generating Models. (June 2022). arXiv:2206.15000 [cs.HC] http://arxiv.org/abs/2206.15000

[9] Kent Beck. 1999. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., USA. https://dl.acm.org/doi/10.5555/318762

[10] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[11] Andrew Begel and Nachiappan Nagappan. 2008. Pair programming: what's in it for me?. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (Kaiserslautern Germany). ACM, New York, NY, USA. https://doi.org/10.1145/1414004.1414026

[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. *Queueing Syst.* 20, 6 (Jan. 2023), 35–57. https://doi.org/10.1145/3582083

[13] Bobby Bodenheimer, B Sanders, M R Kramer, K Viswanath, R Balachandran, Kadira Belynne, and Gautam Biswas. 2009. Construction and Evaluation of Animated Teachable Agents. *J. Educ. Technol. Soc.* (2009). https://www.semanticscholar.org/paper/2899ac4dfe209db4767ec01b5df337079bada517

[14] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. (March 2023). arXiv:2303.12712 [cs.CL] http://arxiv.org/abs/2303.12712

[15] Chen Cao. 2023. Scaffolding CS1 Courses with a Large Language Model-Powered Intelligent Tutoring System. In *Companion Proceedings of the 28th International Conference on Intelligent User Interfaces* (Sydney, NSW, Australia) *(IUI '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 229–232. https://doi.org/10.1145/3581754.3584111

[16] E A Chaparro, Aybala Yuksel, Pablo Romero, and Sallyann Bryant. 2005. Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education. *Annual Workshop of the Psychology of Programming Interest Group* (2005). https://www.semanticscholar.org/paper/c095f0d9b17cd9c2851000534740e7cc087253fa

[17] Jan Chong and Tom Hurlbutt. 2007. The Social Dynamics of Pair Programming. In *29th International Conference on Software Engineering (ICSE'07)*. ieeexplore.ieee.org, 354–363. https://doi.org/10.1109/ICSE.2007.87

[18] Alistair Cockburn and L Williams. 2001. The costs and benefits of pair programming. *Computer Science* (2001). https://www.semanticscholar.org/paper/5ff7b75b20fdbfae23587b660b7093aec2f48e69

[19] Bernardo José da Silva Estácio and Rafael Prikladnicki. 2015. Distributed Pair Programming: A Systematic Literature Review. *Information and Software Technology* 63 (July 2015), 1–10. https://doi.org/10.1016/j.infsof.2015.02.011

[20] Wei Dai, Jionghao Lin, Flora Jin, Tongguang Li, Yi-Shan Tsai, Dragan Gasevic, and Guanliang Chen. 2023. Can Large Language Models Provide Feedback to Students? A Case Study on ChatGPT. (April 2023). https://doi.org/10.35542/osf.io/hcgzj

[21] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, Zhen Ming, and Jiang. 2022. GitHub Copilot AI pair programmer: Asset or Liability? *ArXiv* (2022). https://doi.org/10.48550/ARXIV.2206.15331

[22] 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 *Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/3511861.3511863

[23] Sue Fitzgerald, Renée McCauley, Brian Hanks, Laurie Murphy, Beth Simon, and Carol Zander. 2010. Debugging From the Student Perspective. *IEEE Trans. Educ.* 53, 3 (Aug. 2010), 390–396. https://doi.org/10.1109/TE.2009.2025266

[24] S Freudenberg, Pablo Romero, and Benedict Du Boulay. 2007. Talking the talk: Is intermediate-level conversation the key to the pair programming success story?. In *AGILE 2007*. unknown, 84–91. https://doi.org/10.1109/AGILE.2007.1

[25] Github. [n. d.]. GitHub Copilot Labs. https://githubnext.com/projects/copilot-labs/. https://githubnext.com/projects/copilot-labs/ Accessed: 2023-5-19.

[26] GitHub. 2021. Your AI pair programmer: Copilot. https://github.com/features/copilot. https://github.com/features/copilot Accessed: 2022-10-5.

[27] Google. [n. d.]. Bard. https://bard.google.com/. https://bard.google.com/ Accessed: 2023-5-19.

[28] Keun-Woo Han, Eunkyoung Lee, and Youngjun Lee. 2010. The Impact of a Peer-Learning Agent Based on Pair Programming in a Programming Course. *IEEE Trans. Educ.* 53, 2 (May 2010), 318–327. https://doi.org/10.1109/TE.2009.2019121

[29] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. Pair programming in education: a literature review. *Comput. Sci. Educ.* 21, 2 (June 2011), 135–173. https://doi.org/10.1080/08993408.2011.579808

[30] Jo E Hannay, Erik Arisholm, Harald Engvik, and Dag I K Sjoberg. 2010. Effects of Personality on Pair Programming. *IEEE Trans. Software Eng.* 36, 1 (Jan. 2010), 61–80. https://doi.org/10.1109/TSE.2009.41

[31] 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* 51, 7 (July 2009), 1110–1122. https://doi.org/10.1016/j.infsof.2009.02.001

[32] Steffi Heidig and Geraldine Clarebout. 2011. Do pedagogical agents make a difference to student motivation and learning? *Educational Research Review* 6, 1 (Jan. 2011), 27–54. https://doi.org/10.1016/j.edurev.2010.07.004

[33] Kenneth Holstein, Vincent Aleven, and Nikol Rummel. 2020. A Conceptual Framework for Human–AI Hybrid Adaptivity in Education. *Artificial Intelligence in Education* 12163 (June 2020), 240. https://doi.org/10.1007/978-3-030-52237-7_20

[34] John J Horton. 2023. Large Language Models as Simulated Economic Agents: What Can We Learn from Homo Silicus? (Jan. 2023). arXiv:2301.07543 [econ.GN] http://arxiv.org/abs/2301.07543

[35] Saki Imai. 2022. Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. ieeexplore.ieee.org, 319–321. https://doi.org/10.1145/3510454.3522684

[36] Randall W Jensen. 2005. A Pair Programming Experience. *ACCU - professionalism in programming Overload* 13, 65 (Feb. 2005). https://accu.org/journals/overload/13/65/jensen_254/

[37] Eirini Kalliamvakou. 2022. Research: quantifying GitHub Copilot's impact on developer productivity and happiness. https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/. https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/ Accessed: 2022-10-13.

[38] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2022. Large Language Models are few-shot testers: Exploring LLM-based general bug reproduction. *ArXiv* (2022). https://doi.org/10.48550/ARXIV.2209.11515

[39] 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. (Feb. 2023). arXiv:2302.07427 [cs.HC] http://arxiv.org/abs/2302.07427

[40] Shobhan Kumar, Arun Chauhan, and Pavan Kumar C. 2022. Learning Enhancement Using Question-Answer Generation for e-Book Using Contrastive Fine-Tuned T5. In *Big Data Analytics*. Springer Nature Switzerland, 68–87. https://doi.org/10.1007/978-3-031-24094-2_5

[41] 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* (Yokohama, Japan) *(CHI '21, Article 243)*. Association for Computing Machinery, New York, NY, USA, 1–20. https://doi.org/10.1145/3411764.3445659

[42] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. (April 2023). arXiv:2304.03938 [cs.CY] http://arxiv.org/abs/2304.03938

[43] Mengjun Li and Ayoung Suh. 2021. Machinelike or Humanlike? A Literature Review of Anthropomorphism in AI-Enabled Technology. In *Hawaii International Conference on System Sciences 2021 (HICSS-54)*. https://aisel.aisnet.org/hicss-54/in/ai_based_assistants/5/

[44] Lijia Lin, Robert K Atkinson, Robert M Christopherson, Stacey S Joseph, and Caroline J Harrison. 2013. Animated agents and learning: Does the type of verbal feedback they provide matter? *Comput. Educ.* 67 (Sept. 2013), 239–249. https://doi.org/10.1016/j.compedu.2013.04.017

[45] Kim Man Lui and Keith C C Chan. 2006. Pair programming productivity: Novice–novice vs. expert–expert. *Int. J. Hum. Comput. Stud.* 64, 9 (Sept. 2006), 915–925. https://doi.org/10.1016/j.ijhcs.2006.04.010

[46] Mary Margaret Lusk and Robert K Atkinson. 2007. Animated pedagogical agents: does their degree of embodiment impact learning from static or animated worked examples? *Appl. Cogn. Psychol.* 21, 6 (Sept. 2007), 747–764. https://doi.org/10.1002/acp.1347

[47] Phil Maguire, Rebecca Maguire, Philip Hyland, and Patrick Marshall. 2014. Enhancing collaborative learning using pair programming: Who benefits? *AISHE-J* 6, 2 (June 2014). https://ojs.aishe.org/index.php/aishe-j/article/view/141

[48] Richard E Mayer. 2014. Principles based on social cues in multimedia learning: Personalization, voice, image, and embodiment principles. *The Cambridge handbook of multimedia learning* 16 (2014), 345–370. https://books.google.com/books?hl=en&lr=&id=r3rsAwAAQBAJ&oi=fnd&pg=PA345&ots=iUhQ53T8QY&sig=5tQyKi_f-7aILMxRLuwGTLIix3c

[49] Richard E Mayer and C Scott DaPra. 2012. An embodiment effect in computer-based learning with animated pedagogical agents. *J. Exp. Psychol. Appl.* 18, 3 (Sept. 2012), 239–252. https://doi.org/10.1037/a0028616

[50] Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: A Review of the Literature from an Educational Perspective. *Computer Science Education* 18, 2 (June 2008), 67–92. https://doi.org/10.1080/08993400802114581

[51] 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* (Cincinnati, Kentucky) *(SIGCSE '02)*. Association for Computing Machinery, New York, NY, USA, 38–42. https://doi.org/10.1145/563340.563353

[52] Charlie McDowell, Linda Werner, Heather E Bullock, and Julian Fernald. 2006. Pair programming improves student retention, confidence, and program quality. *Commun. ACM* 49, 8 (Aug. 2006), 90–95. https://doi.org/10.1145/1145287.1145293

[53] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2022. Reading between the lines: Modeling user behavior and costs in AI-assisted programming. *ArXiv* (2022). https://doi.org/10.48550/ARXIV.2210.14306

[54] Ambar Murillo and Sarah D'Angelo. 2023. An Engineering Perspective on Writing Assistants for Productivity and Creative Code. *The Second Workshop on Intelligent and Interactive Writing Assistants* (2023). https://cdn.glitch.global/d058c114-3406-43be-8a3c-d3afff35eda2/paper1_2023.pdf

[55] Laurie Murphy, Sue Fitzgerald, Brian Hanks, and Renée McCauley. 2010. Pair debugging: a transactive discourse analysis. In *Proceedings of the Sixth international workshop on Computing education research* (Aarhus, Denmark) *(ICER '10)*. Association for Computing Machinery,

New York, NY, USA, 51–58. https://doi.org/10.1145/1839594.1839604

[56] Brad A Myers, Amy J Ko, Thomas D LaToza, and Youngseok Yoon. 2016. Programmers are users too: Human-centered methods for improving programming tools. *Computer* 49, 7 (July 2016), 44–52. https://doi.org/10.1109/MC.2016.200

[57] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. 2003. Improving the CS1 experience with pair programming. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education* (Reno Navada USA). ACM, New York, NY, USA. https://doi.org/10.1145/611892.612006

[58] N Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)* (2022). https://doi.org/10.1145/3524842.3528470

[59] Amy Ogan, Samantha Finkelstein, Elijah Mayfield, Claudia D'Adamo, Noboru Matsuda, and Justine Cassell. 2012. "Oh dear stacy!": social interaction, elaboration, and learning with teachable agents. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 39–48. https://doi.org/10.1145/2207676.2207684

[60] Venkata Vinod Kumar Padmanabhuni, Hari Praveen Tadiparthi, and Sagar Madina Muralidhar Yanamadala. 2012. Effective pair programming practice-an experimental study. *Journal of Emerging Trends in Computing and Information Sciences* 3, 4 (2012), 471–479. http://www.agilemethod.csie.ncu.edu.tw/agileMethod/download/2012papers/2012%20Effective%20Pair%20Programming%20Practice-%20An%20Experimental%20Study/Effective%20Pair%20Programming%20Practice-%20An%20Experimental%20Study.pdf

[61] Zachary A Pardos and Shreya Bhandari. 2023. Learning gain differences between ChatGPT and human tutor generated algebra hints. (Feb. 2023). arXiv:2302.06871 [cs.CY] http://arxiv.org/abs/2302.06871

[62] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. (Aug. 2021). arXiv:2108.09293 [cs.CR] http://arxiv.org/abs/2108.09293

[63] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. (Feb. 2023). arXiv:2302.06590 [cs.SE] http://arxiv.org/abs/2302.06590

[64] Michael Perscheid, Benjamin Siegmund, Marcel Taeumel, and Robert Hirschfeld. 2017. Studying the advancement in debugging practice of professional software developers. *Software Quality Journal* 25, 1 (March 2017), 83–110. https://doi.org/10.1007/s11219-015-9294-2

[65] Laura Plonka, Judith Segal, Helen Sharp, and Janet van der Linden. 2011. Collaboration in Pair Programming: Driving and Switching. In *Agile Processes in Software Engineering and Extreme Programming - 12th International Conference, XP 2011, Madrid, Spain, May 10-13, 2011. Proceedings*, Vol. 77. unknown, 43–59. https://doi.org/10.1007/978-3-642-20677-1_4

[66] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. (April 2023). arXiv:2304.02491 [cs.HC] http://arxiv.org/abs/2304.02491

[67] David Preston. 2006. Using collaborative learning research to enhance pair programming pedagogy. *SIGITE Newsl.* 3, 1 (Jan. 2006), 16–21. https://doi.org/10.1145/1113378.1113381

[68] Ben Puryear and Gina Sprint. 2022. Github copilot in the classroom: learning to code with AI assistance. *J. Comput. Sci. Coll.* 38, 1 (Dec. 2022), 37–47. https://dl.acm.org/doi/pdf/10.5555/3575618.3575622

[69] Peter Robe and Sandeep Kaur Kuttal. 2022. Designing PairBuddy—A Conversational Agent for Pair Programming. *ACM Trans. Comput.-Hum. Interact.* 29, 4 (May 2022), 1–44. https://doi.org/10.1145/3498326

[70] 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 Trans. Software Eng.* 37, 4 (July 2011), 509–525. https://doi.org/10.1109/TSE.2010.59

[71] S Sankaranarayanan, S R Kandimalla, S Hasan, and others. 2020. Agent-in-the-loop: Conversational agent support in service of reflection for learning during collaborative programming. *Artif. Intell.* (2020). https://link.springer.com/chapter/10.1007/978-3-030-52240-7_50

[72] Advait Sarkar, Andrew D Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? (Aug. 2022). arXiv:2208.06213 [cs.HC] http://arxiv.org/abs/2208.06213

[73] Noah L Schroeder, Olusola O Adesope, and Rachel Barouch Gilbert. 2013. How Effective are Pedagogical Agents for Learning? A Meta-Analytic Review. *Journal of Educational Computing Research* 49, 1 (July 2013), 1–39. https://doi.org/10.2190/EC.49.1.a

[74] Ben Shneiderman and Pattie Maes. 1997. Direct manipulation vs. interface agents. *Interactions* 4, 6 (Nov. 1997), 42–61. https://doi.org/10.1145/267505.267514

[75] Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. 2012. Understanding the impact of Pair Programming on developers attention: A case study on a large industrial experimentation. In *2012 34th International Conference on Software Engineering (ICSE)* (Zurich). IEEE, 1094–1101. https://doi.org/10.1109/ICSE.2012.6227110

[76] Raymund Sison. 2009. Investigating the Effect of Pair Programming and Software Size on Software Quality and Programmer Productivity. In *2009 16th Asia-Pacific Software Engineering Conference*. 187–193. https://doi.org/10.1109/APSEC.2009.71

[77] Joanna Smith, Joe Tessler, Elliot Kramer, and Calvin Lin. 2012. Using peer review to teach software testing. In *Proceedings of the ninth annual international conference on International computing education research* (Auckland, New Zealand) *(ICER '12)*. Association for Computing Machinery, New York, NY, USA, 93–98. https://doi.org/10.1145/2361276.2361295

[78] Shashank Sonkar, Lucy Liu, Debshila Basu Mallick, and Richard G Baraniuk. 2023. CLASS Meet SPOCK: An Education Tutoring Chatbot based on Learning Science Principles. (May 2023). arXiv:2305.13272 [cs.CL] http://arxiv.org/abs/2305.13272

[79] W Sun and G Marakas. 2009. The True Cost of Pair Programming: Development of a Comprehensive Model and Test. *Americas Conference on Information Systems* (2009). https://www.semanticscholar.org/paper/647fc48650e4f19962c8a6feb87f3bdedde9dd04

[80] Chenhao Tan. 2023. On AI Anthropomorphism - Human-Centered AI - Medium. https://medium.com/human-centered-ai/on-ai-anthropomorphism-abff4cecc5ae. https://medium.com/human-centered-ai/on-ai-anthropomorphism-abff4cecc5ae Accessed: 2023-4-23.

[81] Lynda Thomas, Mark Ratcliffe, and Ann Robertson. 2003. Code warriors and code-a-phobes: a study in attitude and pair programming. *SIGCSE Bull.* 35, 1 (Jan. 2003), 363–367. https://doi.org/10.1145/792548.612007

[82] H Holden Thorp. 2023. ChatGPT is fun, but not an author. *Science* 379, 6630 (Jan. 2023), 313. https://doi.org/10.1126/science.adg7879

[83] Karthikeyan Umapathy and Albert D Ritzhaupt. 2017. A Meta-Analysis of Pair-Programming in Computer Programming Courses: Implications for Educational Practice. *ACM Trans. Comput. Educ.* 17, 4 (Aug. 2017), 1–13. https://doi.org/10.1145/2996201

[84] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22, Article 332)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3491101.3519665

[85] Zichao Wang, Jakob Valdez, Debshila Basu Mallick, and Richard G Baraniuk. 2022. Towards Human-Like Educational Question Generation with Large Language Models. In *Artificial Intelligence in Education*. Springer International Publishing, 153–166. https://doi.org/10.1007/978-3-031-11644-5_13

[86] Laurie Williams and Richard L Upchurch. 2001. In support of student pair-programming. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education* (Charlotte North Carolina USA). ACM, New York, NY, USA. https://doi.org/10.1145/364447.364614

[87] Laurie Williams, Eric Wiebe, Kai Yang, Miriam Ferzli, and Carol Miller. 2002. In support of pair programming in the introductory computer science course. *Comput. Sci. Educ.* 12, 3 (Sept. 2002), 197–212. https://doi.org/10.1076/csed.12.3.197.8618

[88] Dakota Wong, Austin Kothig, and Patrick Lam. 2022. Exploring the Verifiability of Code Generated by GitHub Copilot. *ACM on Programming Languages* (2022). https://www.semanticscholar.org/paper/b5051fedaf17836f6b2a042cc4af4155159795c5

[89] Burak Yetiştiren, Işik Özsoy, and Eray Tüzün. 2022. Assessing the Quality of GitHub Copilot's Code Generation. In *18th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '22)*. https://doi.org/10.1145/3558489.3559072

[90] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (San Diego CA USA). ACM, New York, NY, USA. https://doi.org/10.1145/3520312.3534864