arXiv:2304.02491v1 [cs.HC] 5 Apr 2023

“它居然知道我想要什么，真奇怪”：Copilot 对新手程序员的可用性与交互

JAMES PRATHER, 阿比林基督教大学，美国BRENT N. REEVES,阿比林基督教大学，美国PAUL DENNY,奥克兰大学，新西兰BRETT A. BECKER,都柏林大学学院，爱尔兰JUHO LEINONEN,阿尔托大学，芬兰ANDREW LUXTON-REILLY,奥克兰大学，新西兰GARRETT POWELL,阿比林基督教大学，美国JAMES FINNIE-ANSLEY,奥克兰大学，新西兰EDDIE ANTONIO SANTOS,都柏林大学学院，爱尔兰

深度学习的最新发展催生了代码生成模型，这类模型能够根据自然语言和基于代码的提示词，高准确度地生成源代码。这可能会对课堂产生深远影响：如今，学习编程的初学者可以使用免费的工具，自动获取编程练习和作业的解答建议。然而，目前人们对初学者在实践中如何与这些工具交互知之甚少。我们开展了首项观察性研究，考察入门阶段的学生如何在一项典型的编程入门课程（计算机科学导论课程）作业中使用此类代码自动生成工具 GitHub Copilot。通过观察和访谈，我们探讨了学生对这项技术在学习方面优势与不足的看法，呈现了新观察到的交互模式，并讨论学生面临的认知困难和元认知困难。我们结合这些发现探讨了设计启示，具体而言，如何改进 Copilot 等工具，为新手的编程体验提供更好的支持与引导。

CCS 概念：• **以人为中心的计算** → **人机交互（HCI）**；**人机交互实证研究**；**用户研究**；**自然语言界面**；**用户界面编程**；• **计算方法** → **人工智能**；•**社会与专业议题** → **计算教育**；**计算机科学教育**；**计算机科学导论课程**；• **应用计算** → **教育**。

附加关键词和短语：人工智能；人工智能；自动代码生成；Codex；Copilot；计算机科学导论课程；GitHub；GPT-3；人机交互；编程入门；新手编程；OpenAI

ACM 参考格式：
James Prather、BrentN. Reeves、Paul Denny、Brett A. Becker、Juho Leinonen、AndrewLuxton-Reilly、GarrettPowell、JamesFinnie- Ansley 和 Eddie Antonio Santos。2023。“它竟然知道我想要什么，这很奇怪”：Copilot 面向新手程序员的可用性与交互。载于*ACM*《计算机与人类交互汇刊》（*TOCHI*）论文集。ACM，纽约，美国，26 页。https://doi.org/11.1111/1111111.1111111

允许为个人或课堂使用而免费制作本作品的部分或全部内容的数字或纸质副本，前提是这些副本不用于或分发以谋取利润或商业利益，且副本须包含本声明及首页上的完整引文。必须尊重本作品中第三方组件的版权。其他用途请联系所有者/作者。© 2023 版权归所有者/作者所有。稿件已提交给 ACM

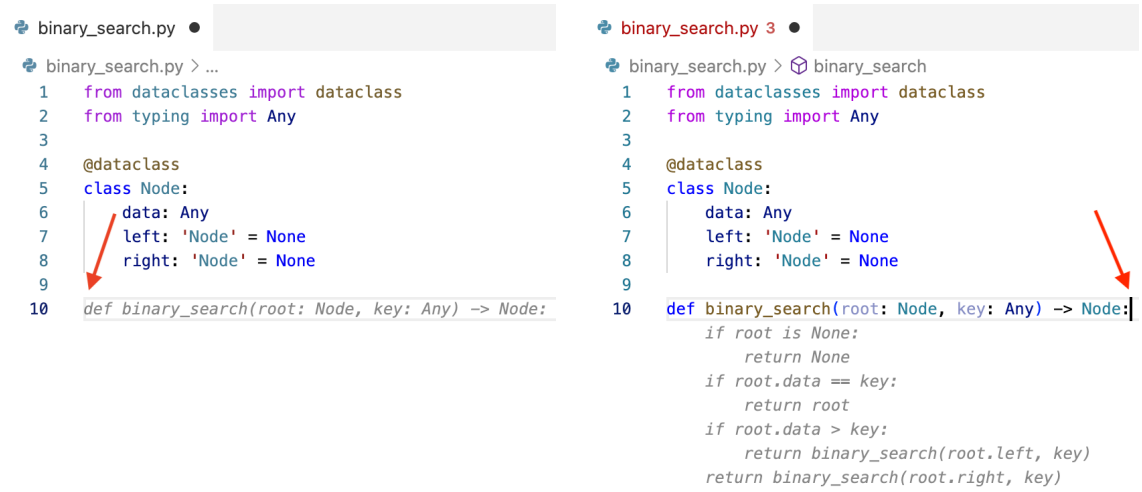
1 引言

编程入门课程通常要求学生编写许多程序 [4]。教师设计编程练习，以促进并提升学生的学习；然而，学生并不总能恰当地关注自己的学习，往往尽可能快地完成任务。因此，在这些编程练习中，教师和学生的需求常常相互冲突。当学生发现自己陷入困境、无法完成所需任务时，这些不同的需求便交汇在一起。大量证据表明，学生难以制定有效计划 [77] ，也难以落实已制定的计划 [28, 42]。这种经历令学生倍感挫折 [10] ，可能阻碍他们的学习进展，并导致抄袭等不良行为 [38]。为了保持学习进度并带来积极的学习体验——这对教师和学生双方都有益——有必要为陷入困境的学生提供支持 [60]。

遗憾的是，教师并非总能提供这类支持。教程、指南、教科书等静态学习资源，以及集成开发环境等工具，都无法提供结合具体情境的交互式支持。人们曾多次尝试为正在学习编程的学生提供更贴合具体情境的帮助，但要有效支持编程仍然是一项挑战 [43]。智能辅导系统 [25]会根据学生的表现提供自适应反馈，但这类系统通常会引导学生完成预先构建的任务，无法在更一般的（真实场景）环境中支持学生学习。近年来，自动提示生成系统已被用于为各类编程练习生成提示 [58]。虽然这类系统无需人工设计针对任务的反馈，但必须部署在能够获取学生历史表现数据的环境中，才能判断针对特定练习哪些方法更有效 [55], ，这限制了它们的实用性 [43]。目前的自动化方法都无法针对学生正在解决的任意编程问题提供情境化支持。学生（及其教师）可以从标准集成开发环境中的自动化支持中受益，尤其是这类支持能够与他们正在处理的问题相关联。大语言模型（LLM）或许能够满足这一需求。

本文从教学与学习的角度，探讨学生在完成编程任务时如何使用大语言模型工具。我们研究了 GitHub Copilot 的使用情况；它是一款由大语言模型 Codex 驱动的集成开发环境插件。Copilot 对初学者而言易于获取，学生可以免费使用，并且可作为插件直接运行于常见的开发环境中。现有研究尚未探讨学生如何与基于大语言模型的工具互动，以助力他们在编程教育中的进步。本文通过开展首项研究，拓展了这一领域的认识：观察大学一年级学生在编程入门课程（通常统称为“计算机科学导论课程” [11]）中，使用 Copilot 完成典型作业的情况。我们尤其关注学生首次使用新工具时的新颖体验。我们将对新手程序员的观察与访谈相结合，探讨他们对这项技术在学习中的益处与风险的看法。

我们发现，大多数学生认为 Copilot 能帮助他们更快地编写代码，同时也担心自己无法理解自动生成代码，并会变得依赖这些工具——教育工作者也有同样的担忧 [22,34]。我们观察到两种新的交互模式。第一种是学生利用 Copilot 自动生成的代码提示来引导它，引导它找到解决方案，而不是专注于从头编写代码并整合 Copilot 的建议。第二种是一些学生被 Copilot 的错误建议带着走，逐渐偏离方向，从一个错误建议转向下一个，最终迷失方向。我们还观察到，学生在使用该工具时面临认知困难和元认知困难。最后，我们基于这些发现提出了伦理考量和设计指南。



（a）程序员在 Copilot 仅提供少量帮助的情况下，已将代码输入至第 10 行行首的光标处（红色箭头）。灰色斜体代码是 Copilot 的建议。

(b) 程序员通过按下 Tab 键接受了 (a) 中的建议，光标随之移至第 10 行末尾（红色箭头处），恰好位于建议代码之后。灰色斜体代码是 Copilot 接下来的建议。

图 1. Copilot 向程序员提供代码建议。更多视频请访问 github.com/features/copilot。

1.1 背景

2021年，OpenAI 发布了 Codex [22]，，这是一个基于公开 GitHub 仓库中数十亿行 Python 代码训练的 GPT-3 版本。[1] Codex 同时熟悉自然语言和编程语言。它最擅长英语和 Python，但也能使用西班牙语等其他自然语言，以及 Go、JavaScript、Perl、PHP、Ruby、Shell、Swift 和 TypeScript 等编程语言。[1]

GitHub Copilot 使用 Codex，根据用户已输入的代码实时提供代码建议。[2] Copilot 于 2022年6月 结束技术预览阶段，现在作为插件免费提供给学生使用，支持 Visual Studio Code 和 JetBrains 等集成开发环境。Copilot 的宣传语是“您的 AI 结对程序员”[2]——这是有意呼应一种广为人知的编程教育方法：结对编程 [62]。图 1 展示了 Copilot 如何在程序员编写代码时实时提供建议。

本研究聚焦于 Copilot，不过还有其他几种 AI 代码生成器可供使用（详见第 2.1.1 节）。我们的工作旨在分析新手程序员如何使用 Copilot，并通过访谈了解新手程序员使用 Copilot 的体验。

1.2 研究问题与贡献

我们的研究问题如下：

RQ1: 初学者初次接触 GitHub Copilot 时会如何与之互动？**RQ2:** 初学者如何看待自己首次使用 GitHub Copilot 的体验？

本研究的创新贡献如下：

[1]openai.com/blog/openai-codex [2] github.com/features/copilot

（1）我们首次探索了初学者在本科编程入门课程中，使用 Copilot 完成典型学习作业的情况。我们还首次访谈了此前从未使用过 Copilot（或类似工具）的初学者，了解他们初次使用 Copilot 的体验，以探究他们认为这类工具有哪些益处，以及他们对此有哪些顾虑。虽然已有一些研究探讨 Copilot 在解决初学者编程问题方面的能力 [34], 但尚无研究关注该工具对初学者的可用性，也没有探究他们对它的看法。此外，未来可能无法再收集到这类反应，因为届时要找到一个全班学生都从未接触过人工智能代码生成器的班级，很可能并不现实。（2）我们提出了使用基于大语言模型的代码生成工具的全新交互模式：漂移和引导。这两种模式补充了相关文献中已有的大语言模型交互模式，例如探索和加速。（3）我们探讨了初学者使用 Copilot 等人工智能代码生成器时的体验，并提出了四项设计启示。

2 相关工作

本节回顾了大语言模型、它们在计算领域教育中的应用，以及此前关于 AI 代码生成器的用户研究等相关近期工作。

2.1 大语言模型

近年来，自然语言处理领域的大语言模型（LLM）取得了长足进展。这类模型通常基于深度学习 Transformer 架构，并且在大多数生成任务中往往超越此前最先进的模型。例如，GPT-3（一种文本到文本模型）能够生成难以与人类撰写的文本区分的内容 [19], ，其应用包括摘要、翻译、回答问题以及各种其他文本任务。

大语言模型近期性能的提升，主要得益于使用海量数据进行训练以及模型规模的扩大。例如，GPT-3 使用 45 太字节的文本数据进行训练，拥有 1750 亿个模型参数 [19]。近期大语言模型规模的扩大带来了“涌现能力”——较小模型未曾展现、似乎仅因模型规模增大而产生的能力 [83]。大语言模型通常由其开发者进行预训练，之后再向他人开放模型访问权限。模型通过“提示词”接收输入；提示词是模型尝试完成的自然语言片段。对大多数用户而言，模型的内部架构并不透明，因此人们采取了多种方法来构建有效的提示词，这些方法通常称为“提示工程” [51]。关于提示工程的详细说明，请参见 [51]。

2.1.1 人工智能代码生成。 除了文本转文本和文本转图像模型（如 Dall·E 23）之外，近期还发布了几种专门用于生成编程源代码的模型。这些模型包括 DeepMind AlphaCode [49], Amazon CodeWhisperer [5], CodeBert [33],Code4Me,[4] FauxPilot,[5] 和 Tabnine.[6] 这些模型要么使用源代码进行训练，要么是在大语言模型的基础上，以源代码形式的额外训练数据进行增强。虽然其中大多数面向专业人士，但 Copilot 的使用门槛较低，初学者也能轻松上手，而且学生可以免费使用。

这些模型在生成可运行代码方面展现出了出乎意料的能力。DeepMind 声称，AlphaCode 在编程竞赛中的表现可与中位数水平的参赛者相当 [49]。Finnie-Ansley 等人

[3]openai.com/dall-e-2/ [4]
code4me.me[5]
github.com/moyix/fauxpilot [6]
tabnine.com

发现，Codex 解决编程入门问题的能力胜过平均水平的学生；在给出相同的编程入门考试题目时，其表现位于真实学生的前四分位数 [34]。Chen 等人发现，当模型收到生成测试用例的提示词，并使用这些测试用例来筛选生成的源代码时，根据自然语言输入生成正确源代码的表现有所提升 [21]。

除了最初用于生成源代码的目的外，这类模型还被发现能够胜任其他任务。例如，Pearce 等人探索了使用多个模型修复代码漏洞。尽管他们发现这些模型能够修复 100% 的合成示例漏洞，但在真实案例中的表现并不理想 [66]。Pearce 等人的另一项研究考察了人工智能代码生成器在逆向工程中的适用性 [67]。使用源代码训练的大语言模型也擅长解决其他问题，例如大学水平的数学问题 [30], 概率与统计 [79], 以及机器学习问题 [85]。

2.2 AI代码生成器与计算教育

迄今为止，计算机教育文献中对人工智能代码生成器的评估寥寥无几。鉴于这类工具刚刚问世，它们将对教育实践产生何种影响，目前仍不清楚。不过，研究人员已开始表达对其使用的担忧。在探讨这类模型带来的机遇与风险的研究中，Bommasani 等人明确将 Copilot 列为教育工作者面临的一项挑战 [14], , 并指出，如果学生开始过度依赖它，可能会对学习产生负面影响。他们还担心，难以判断某个程序究竟是由学生编写，还是由 Copilot 之类的工具生成。Chen 等人在介绍 Codex 的论文中也提出了类似的担忧，认为人们可能会过度依赖这类工具 [22]。他们将“探索强大的代码生成技术可用后，程序员的教育……历程可能受到影响的各种方式”列为未来研究方向 [22]。学生将如何采用并使用 Copilot 之类的工具，目前尚不明确 [32], , 但可以肯定的是，这类工具将在课堂内外发挥日益重要的作用。

就计算教育领域的实证研究而言，人工智能代码生成器的评估主要围绕其解决编程入门问题的表现，以及生成新问题的能力展开。Finnie-Ansley 等人的早期研究考察了 Codex 解决典型编程入门问题（取自其所在机构的考试题目）的表现，以及解决若干“降雨量”问题的常见（且模型未见过的）变体的能力 [34]。最终，该模型在两次测试中的得分约为 80%；将其表现与选修该课程的学生进行比较时，排名为第 17 位（共 71 人）。此外，在“降雨量”任务中，Codex 能够生成多个正确解答，这些解答在算法方法和代码长度方面均有所不同。然而，这项研究中的问题总体上相当简单，对于更复杂的问题，可能需要人类与模型进行更多互动 [6]。

最近，Sarsa 等人探索了 Codex 的自然语言生成能力，利用它来合成适合编程入门课程的新颖编程练习和代码解释 [75]。他们通过向模型提供单个示例练习作为输入（“一次性”学习）来生成编程练习，并试图创建针对类似概念但涉及规定主题的新问题。他们发现，远超 80% 的生成练习包含可执行的示例代码解答，但该代码仅在 30% 的情况下通过了同样由 Codex 生成的测试用例。此外，大约 80% 的练习包含使用规定主题的自然语言问题描述，这展示了模型轻松自定义输出的能力。此外，他们使用 Codex 生成通常在编程入门课程中看到的代码样本的自然语言解释。对解释的彻底性以及种类的

存在的错误揭示，在 90% 的情况下代码的所有部分都得到了解释，但只有 70% 的单独行具有正确的解释。

2.3 人工智能代码生成工具的用户研究

近期研究探讨了开发者如何使用 Copilot 等代码生成 IDE 插件。Vaithilingam 等人让参与者在 VS Code 中使用 Python 完成三项编程任务 [80]。其中一项任务使用了 Copilot；在其他情况下，他们使用的是 VS Code 内置的 IntelliSense 代码补全。尽管参与者使用 Copilot 并未节省时间，但多数参与者（19/24）更偏好 Copilot 而非 IntelliSense，并将“节省时间”列为其优点。参与者对 Copilot 的正面评价包括它能生成起始代码，并为程序员提供一个起点，即使这些起始代码可能让他们陷入“调试泥潭”。另一方面，一些参与者认为 Copilot 生成的代码难以理解且不可靠。与本研究聚焦于新手程序员不同，Vaithilingam 等人的研究中，编程经验少于两年的参与者只有一人。Jayagopal 等人观察了新手程序员使用五种程序合成工具的情况 [40]。他们发现，与要求提供更正式规范的工具相比，能够通过在编辑器中开始编写所需代码来向工具输入提示词，“似乎更令人兴奋，也不那么令人困惑”。他们还观察到，如果生成的代码不符合规范，初学者会进行提示工程，有时也会仔细阅读代码建议。Barke 等人提出了关于与 Copilot 交互的扎根理论。他们邀请 20 名参与者（其中 9 人已有使用 Copilot 的经验）完成任务，并发现交互可分为“加速模式”和“探索模式”：在加速模式下，程序员使用 Copilot 补全自己已计划编写的代码；在探索模式下，程序员提示 Copilot 编写自己不确定如何完成的代码 [8]。他们观察到，过度依赖 Copilot 可能导致任务完成率下降，而需要从多条建议中进行选择则可能造成认知过载。

3 研究方法

为了更好地了解初学编程的学生如何与人工智能代码生成器互动，我们开展了一项研究，观察参与者使用 GitHub Copilot 的情况。随后，我们采访了学生，了解他们的使用体验。

3.1 参与者与背景

参与者均为18至22岁的大学生，就读于美国一所中西部私立研究型大学的编程入门（计算机科学导论课程）课程。授课语言为 C++。我们招募了19名学生参与研究，其中5人认同自己为女性，14人认同自己为男性。所有参与者都是新手程序员，入学时几乎没有或完全没有编程经验。研究于2022年4月春季学期最后一周进行。此前，所有参与者都没有接触过 Copilot；研究开始前，我们简要介绍了预期情况以及如何使用它。

我们观察了学生以与该学期其他所有作业相似的方式完成一项新的家庭作业。该学期的每项作业都通过一个插件显示在 Canvas学习管理系统 中，该插件连接到我们的自动化评估工具。这项作业（见图 2）以经典游戏“扫雷”为蓝本，难度相当于两周前就可能布置的程序设计作业。解决这个问题需要接收输入、使用嵌套循环、检查二维存储是否满足特定条件、在满足这些条件后更新接收到的数据，并输出结果。与该学期的所有编程作业一样，学生可以先查看题目说明，再进行编码并提交解答。观察在上课时间于常规教室旁边的一间教室中进行。这样，研究情境与该学期其他监考的课堂编程作业相似

，学生在该学期收到的编程作业也是如此。唯一的区别是，每次只观察一名学生。

3.2 流程

一名研究人员独自在房间里，每次观察一名学生，并按照出声思维协议 [31]，记录他们的行为和言语。每位参与者有 30 分钟完成程序，并可以使用 Copilot 以及笔记或互联网等任何资源。这与该学期其他受监考的课堂代码编写活动的时限相同。

每位参与者都使用启用了 Copilot 的 Visual Studio Code（VS Code）。Copilot 会以浅灰色文字建议一行或多行代码（图 1），学生可以按 Tab 键接受，也可以按 Escape 键拒绝。建议出现时，学生也可以继续输入。学生输入时，Copilot 会立即根据文本文件中已有的内容开始建议代码；随着代码逐渐增多，这些建议通常也会变得更准确、更有用，且与任务更相关。

完成程序后，或规定时间结束后，我们进行了简短访谈，并对访谈内容进行了人工转录。我们的访谈问题如下：

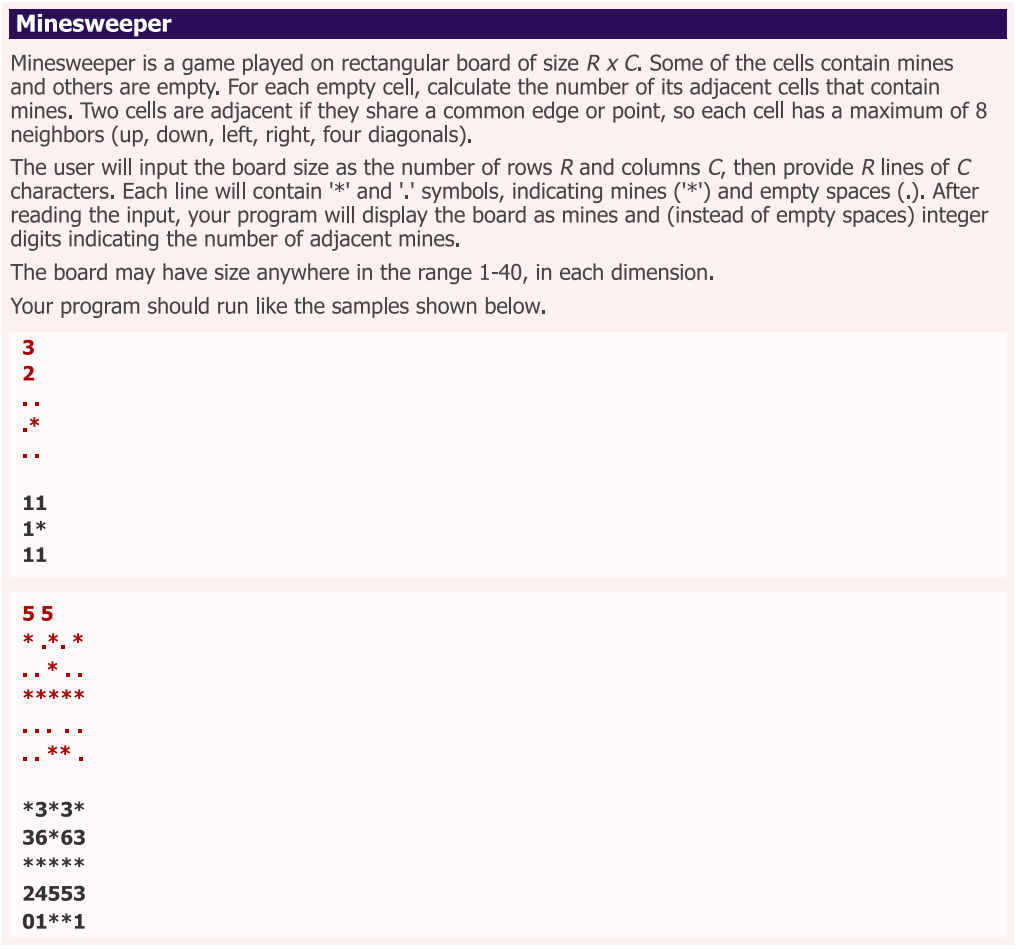
（1）你认为 Copilot 帮助你更好地理解了如何解决这个问题吗？如果是，为什么？如果不是，为什么？（2）如果你有这样一个工具，而且教师允许使用，你认为自己会在编程作业中使用它吗？如果会，如何使用？（3）你认为 Copilot 这样的工具有哪些优点？（4）你对 Copilot 这样的工具有哪些顾虑或担忧？

尽管访谈问题通常更为开放，但我们采用了这些有针对性的问题，以聚焦于以用户为中心的方法，了解作为用户的学生希望这类系统具备什么。我们尝试以这样的方式设计问题，使学生的回答对教师有价值且值得关注。

3.3 分析

为分析数据，我们采用了反思性主题分析 [15, 17]，旨在理解定性数据中呈现的模式和意义。这种方法要求研究者深入参与数据分析，并以灵活、探索且循环往复的过程形成主题。随着研究者对数据愈发熟悉，编码会不断变化，研究者也会通过迭代过程合并和拆分编码；这一过程贯穿整个分析 [16]。由于编码是在分析过程中逐步形成的，因此不计算编码者间信度，而是通过其他方式确保结果的可靠性。为确保分析的可靠性，随着主题逐渐浮现，作者们举行了数次小组会议，对比编码并讨论差异，这符合反思性主题分析的做法 [61]。我们的主题是数据分析过程的产物，而非某些其他定性分析方法中的输入。按照 Braun and Clark [15]的概述，我们在分析中采用了六个主要阶段：

（1）熟悉数据：观察笔记和对访谈问题的回答与研究团队共享。研究团队中的四名成员专注于分析，阅读部分数据以熟悉数据，并记录初步想法。（2）生成初始编码：参与分析的研究人员开会讨论初始编码及体现这些编码的数据示例。随着对数据逐步形成共同理解，这些编码在四次会议中经过合并、拆分和扩展。



Minesweeper

Minesweeper is a game played on rectangular board of size *R x C*. Some of the cells contain mines and others are empty. For each empty cell, calculate the number of its adjacent cells that contain mines. Two cells are adjacent if they share a common edge or point, so each cell has a maximum of 8 neighbors (up, down, left, right, four diagonals).

The user will input the board size as the number of rows *R* and columns *C*, then provide *R* lines of *C* characters. Each line will contain '*' and '.' symbols, indicating mines ('*') and empty spaces (.). After reading the input, your program will display the board as mines and (instead of empty spaces) integer digits indicating the number of adjacent mines.

The board may have size anywhere in the range 1-40, in each dimension.

Your program should run like the samples shown below.

```
3
2
..
.*
..

11
1*
11
```

```
5 5
*.*.*
..*..
*****
... ..
..**.

*3*3*
36*63
*****
24553
01**1
```

Fig. 2. 自动化评估工具给出的提示词，描述了参与者被要求解决的问题 e.

（3）寻找主题：随后将这些编码归为潜在主题，并进行了深入讨论。研究人员找出了体现这些潜在主题的数据示例。（4）审查主题：结合整个数据集对主题进行审查，并通过讨论和重新整理编码加以完善，使其更能反映数据。（5）界定并命名主题：为主题拟定名称并撰写清晰的描述，确保其与数据及研究的整体背景保持一致。（6）撰写：分析在撰写过程中完成，并建立了研究问题、相关文献、主题与数据之间的联系。

我们将饱和定义为“新增数据不再产生任何新的主题”的时点 [36]。由于访谈请参与者回顾他们在观察研究中的经历，因此我们将观察和访谈所得的数据视为同一个数据集。由此，对完整数据集进行分析后得出的主题，构成了关于参与者使用 Copilot 经历的连贯叙述。

本研究所采用的所有方法以及收集的所有数据，均已获得 [匿名机构]的机构审查委员会批准。所有参与者在参与前均签署了知情同意书，了解本研究的相关情况。收集参与者的数据后，我们立即对其进行匿名化处理，并创建了一份仅收集数据的研究人员可访问的密钥。其他研究人员看到的均为完全去标识化的数据。所有数据均安全存储在受多因素身份验证保护的 Google 云端硬盘中。

表1. 根据观察和访谈归纳的主题与子主题。

	观察			访谈		
主题	子主题	数量	独特的	子主题	数量	独特
交互	编码	244	19	编码	11	5
	接受	93	19	接受	10	4
	UX	68	17	UX	33	14
	拒绝	64	17	拒绝	3	3
认知	困惑	65	13	困惑	11	8
	积极情绪	30	6	积极情绪	8	5
	元认知	24	10	元认知	17	10
	消极情绪	6	4	消极情绪	14	9
目的	引导	4	3	引导	49	15
	外包	3	3	外包	46	17
	速度	1	1	速度	37	17
推测	智能	6	3	智能	9	4
	未来	2	2	未来	13	8
	没有缺点	0	0	没有缺点	10	8

4 结果

我们从观察和访谈数据中识别出几个主题：**交互**、**认知**、**目的**和**推测**。表1列出了这些主题及其包含的子主题，并注明了各子主题在数据集中出现的次数，以及我们观察到有多少位不同参与者做出或说出与该子主题相关的内容。子主题“没有缺点”仅出现在访谈中，未出现在观察中。表 2 还按访谈问题列出了各子主题层面的回答情况。本节其余部分按主题整理并综合观察和访谈的结果，辅以具有代表性的引述，以说明相关主题并展现参与者的思考。

4.1 主题：交互

这一主题涵盖了参与者完成任务时的交互与行为观察，以及观察记录中的话语和访谈中的反思，这些内容都体现了相关交互。它是数据集中最常见的主题，包含四个子主题。**编码** 由我们观察到的三类行为组成：编码活动、适配自动生成的代码，以及解读 Copilot 的建议。我们用“编码活动”指代未在其他地方得到更具体归类的某种编程相关任务。**接受** 在此指 Copilot 生成建议后按下 Tab 键，从而将建议的代码插入文件。**用户体验** 旨在区分 Copilot 在用户看来呈现的样貌



表 2. 子 -访谈研究中出现的主题，按访谈问题列出（包括数量超过 4 的项目）。

问题	访谈 子主题	数量	唯一
你认为 Copilot 是否帮助你更好地理解 如何解决这个问题？如果是，为什么？如果不是，为什么？	引导	20	10
	UX	14	7
	外包	12	8
	元认知	9	8
	速度	9	8
	困惑	8	6
	编码	7	4
	智能	6	3
	积极情绪	5	3
如果你自己有这样一个工具，而且老师允许使用， 你觉得自己会用它来完成 编程作业吗？如果会，你会怎么用？	速度	13	11
	外包	10	7
	引导	8	5
你认为像 Copilot 这样的工具有哪些优势？	引导	20	10
	速度	15	15
	外包	7	6
对于像这样的工具，你有什么担忧或顾虑？ Copilot？	外包	17	12
	未来	10	7
	UX	9	4
	没有缺点	6	6

（即用户界面）以及使用过程中遇到的困难（即可用性）。从标记为这一主题的数据中，我们发现了几种有趣的交互模式，下面将对此进行讨论。最后，当参与者看到 Copilot 的建议后，却继续输入与该建议无关的内容时，就视为**拒绝**。

4.1.1 交互模式：引导。
我们注意到的第一种交互模式是“慢速接受”现象：参与者会逐字逐句地输入 Copilot 的建议，却不会直接接受它（即按下Tab 键）。这种情况并不只发生在他们首次遇到 Copilot 建议时。有一次，参[redacted]者 #4 先通过慢速输入接受了一个 for 循环的建议，随后按下Tab 接受了 Copilot 对循环体内容的建议，之后又在下一个循环中进行了慢速接受。参与者 #9 在[redacted]话接近尾声时进行了一次慢速接受；此前，他们已经多次直接接受、拒绝和调整建议。这可能表明，新手程序员不确定是否要将大量并非由自己编写的代码放入文件中。参与者 #1 说：

> P01：“我大部分时间都在解读它给出的代码.....如果我看到一个自己大致理解的提示词，我可能会用它来帮助自动填充一些小部分。我可能会浏览一大段代码，看看它是否确实适合我，以及这是不是我想采用的做法。对于不太熟悉这门编程语言的人来说，这可能会成为阻碍。你可能有一段能运行的代码，却完全不知道它是如何工作的。”

我们注意到的第二种交互行为是“回溯”。当参与者删除刚刚接受、且未做任何修改的代码时，就会出现这种情况。共有 13 位参与者至少这样做过一次，这是我们观察到的出现频率第五高的行为。这表明，新手程序员可能会在未仔细审阅自动生成的建议前就接受它们，随后又立即将其删除，从而导致工作流程受到干扰。参与者的一些引述进一步说明了这一点：

P06：“它会给你不知道自己是否需要的代码，这会让你更加困惑。”

P10：“一个缺点是，你必须阅读建议，然后再把它删除。”

P14：“当我按下 Tab 键，而它完全不是我需要的内容时，我觉得很烦人。”

我们注意到的第三种交互行为是“适配”。新手程序员通常会直接接受为他们生成的代码，但也会对其进行适配以满足自己的需求。许多参与者将大部分时间都花在适配为他们生成的代码上，而很少从头开始编写自己的代码。这可能促成了他们认为 Copilot 为自己节省了时间的感受。这些行为（慢速接受、回溯和适配）促成了第一个新颖的大语言模型交互模式，即引导，其理念是：学生将大部分时间用于尝试促使工具（即 Copilot）生成他们认为自己需要的代码。

4.1.2 交互模式：漂移。

经常观察到参与者对 Copilot 生成的错误代码进行适配。换言之，适配格式错误的自动生成代码，只会让他们在“调试泥潭”中越陷越深，离正确解法越来越远；这一现象在先前的一项 Copilot 用户研究中也曾被观察到 [80]。与此相反，观察发现，参与者 #13 多次尝试解读代码，却从未进行适配。他们接受了 Copilot 的许多建议，即使这些代码建议并不能帮助他们解决问题。尽管如此，在编码环节接近尾声时，他们这样说道：

P13：“感觉它好像能生成我正在想的东西。总觉得不太对劲，你知道吧？”

最后，理解参与者与自动生成的建议互动的方式也很重要。当 Copilot 生成错误代码时，就会出现一个可用性问题。然而，更常见的情况是，Copilot 经常建议一大段代码，这似乎让参与者感到厌烦。建议代码不断涌现也会分散注意力，因为大多数参与者会在建议代码出现后尝试阅读它（表现出解读行为）。这表明，当 Copilot 不断打断初学者的问题解决过程时，他们可能难以有效使用它。参与者以各种方式应对这一问题，并表示：

P01：“在你不需要时，它还一直提示内容，导致很难看清自己正在输入什么。” P06：“我不需要时，它还一直提示内容。” P10：“有些建议太长，让我困惑它实际建议的是什么。花时间阅读，而不是思考，浪费了时间。” P15：“怎样才能让它只生成一行，而不是整个内容？”

Copilot 旨在作为帮助开发者的工具。我们观察到的交互行为，并不能反映资深开发者如何使用 Copilot。Barke 等人报告称，资深程序员会用它加快工作进度（加速）或探索可选方案（探索） [8]。尽管我们也注意到了上述两种交互模式，但基于前文的讨论，我们还提出了第二种新颖的交互模式：漂移。看到一条建议后，用户慢慢接受它、对其进行适配、将其删除，然后重复这一循环。用户在一条条建议之间漫无目的地漂移，没有明确方向，任由 Copilot 推着自己向前。

4.2 主题：认知

该主题包含反映参与者认知状态（即他们当时的想法或感受）的观察和话语，并由四个子主题构成。**困惑** 指参与者不理解 Copilot 生成的代码的情况

、对 Copilot 本身的工作方式感到困惑，以及其他与 Copilot 无关的相关因素。**元认知** 是对思考过程的思考。在编程情境中，它涉及程序员如何解决问题，以及他们为此经历的过程 [53]，编程环境也可能在其中发挥作用 [41]。尽管这是一种难以观察的现象，但我们观察到参与者在提示词上遇到困难并反复阅读提示词、用纸笔推演问题，以及在卡住时使用 Copilot探索可能的解决方案。**积极情绪** 指参与者在观察过程中口头表达某种积极情绪或感叹的情况，包括笑声、惊叹和兴奋。许多参与者确实对 Copilot 的能力感到惊讶和高兴。**消极情绪** 同样指参与者在编程任务过程中感到沮丧或恼怒的情况。这通常与 Copilot 给出的建议中出现大段代码有关。不过，建议不正确或无帮助时也会出现这种情况。

4.2.1 寻找前进之路。

参与者常常会被 Copilot 生成的输出弄糊涂。我们还观察到，Copilot 自动生成的反馈并不总是正确，尤其是在使用初期。例如，参与者 #9 看到 Copilot 针对与问题规格说明不符的输入生成建议，便大声问道：“这样对吗？”参与者 #14 在 Copilot 生成评论而非一行代码时口头表达了困惑，这使他们改变了对 Copilot 会做什么以及如何使用它的理解。参与者 #10 与 Copilot 交互了几分钟，接受了多条建议，但其行为和言谈仍表明，他们认为 Copilot 也会检查程序是否正确（很像自动化评估工具）。以下引述说明了初学者使用 Copilot 时遇到的认知困难：

> P6：“如果你不知道自己在做什么，它可能会让你更加困惑。它给出的代码，你根本不知道是做什么的。”
> P13：“ 它很碍事，会打乱我的思路。我正在想解决方案，但它不断用建议填满屏幕。我会忍不住照着它说的做，而不是自己思考。”

然而，其他参与者能够在卡住时使用 Copilot 的建议，作为找到前进路径的一种方式。使用像 Copilot 这样的系统在卡住时发现解决方案，在以下引述中得到了完美的说明：

> P15：“这有点像没有人的橡皮鸭调试法。”

“橡皮鸭调试法”是一些程序员使用的一种做法，涉及向某人或某物（经典的是橡皮鸭）口头表达问题。这位参与者理解橡皮鸭调试法的价值，似乎表明通常更喜欢与人而不是无生命物体一起练习。更有趣的是，P15 似乎认为 Copilot 可以替代这种做法，这意味着它可以充当元认知支架（即促进思考他们试图解决的问题以及他们在问题解决过程中的位置）。

4.2.2 情绪。

使用 Copilot 的认知和元认知层面引发了参与者积极和消极的情感。例如，参与者 #1 看到 Copilot 生成的第一条多行建议时笑了起来。参与者 #14 在实验过程中不同时间说道：“哇，这也太奇怪了”和“这简直不可思议！”许多参与者流露出的情绪似乎与喜悦和惊讶有关。看到 Copilot 生成的代码后，参与者 #18 说：“哦！这挺酷的！它好像读懂了我的心思！”，“哦，哇。等等。这太疯狂了。”以及“你是从哪儿找到这个的？我学习编程的时候怎么没有这个？”虽然只有约三分之一的参与者表达了积极

情绪，但这表明，当初次体验顺利如愿时，可能会产生这类情绪反应；我们将在设计启示部分进一步讨论这一点。积极情绪的益处不止于令人兴奋，还能直接帮助那些可能因学习编程而感到畏惧或焦虑的学生：

> P3：“对于像我这样不知道自己在做什么、没有任何经验就开始编码的人来说，一旦走上正确的道路，看到那些能帮助自己的建议，就会更有动力。这让我觉得自己更清楚自己在做什么，对自己的工作也更有信心，并且相信自己能继续在计算机科学领域发展职业生涯。”

另一方面，负面情绪会降低学生坚持完成学习过程中所需任务的能力 [44]。这可能是花了很长时间仍无法在问题解决过程中取得进展这样的大问题，也可能是收到编译器以错误消息形式给出的反馈这样的小挫折。在我们的观察中，有一个例子是参与者 #15 不小心接受了 Copilot 建议的一些代码，随后明显因添加了这些代码而感到烦躁。一些参与者似乎对 Copilot 要求他们投入过多注意力感到沮丧。例如，参与者 #7 表示：“它总是自动填入我不想要的内容。”类似地，参与者 #15 对 Copilot 建议的大段代码感到恼火，并表示：“它给出的代码片段太大，让人很沮丧。”尽管这类负面交互相对少见，但尚不清楚随着长期使用 Copilot，这类交互会减少还是加剧。

4.3 主题：目的

这一主题概述了参与者使用 Copilot 的原因，以及这些动机可能引发的问题；相关内容来自观察期间的话语和访谈中的反思。该主题由三个不同的子主题构成。**引导** 指 Copilot 在编程问题解决过程中为参与者提供帮助，例如帮助他们学习新知识，或发现此前未知的边界情况。**外包** 包含两个不同的观点。其一，参与者可能担心 Copilot 生成可运行但他们无法理解的代码，因此可能开始把它视为“黑箱”。其二，参与者担心 Copilot 可能成为一根拐杖。最后，**速度** 指参与者认为 Copilot 能帮助他们比独立完成时更快地完成作业。

4.3.1 学习。

参与者对 Copilot 提供的引导给予了积极评价。这类评论通常涉及较高层次的方向指引和问题解决。例如：

> P05：“如果我大致知道该怎么做，它或许能帮我完成，或者让我知道接下来该做什么。” P06：“或许能提供一些解决问题的有用思路。” P15：“我大概知道自己需要做什么，只是不知道如何实现。” P19：“它正在引导我找到答案。”

有三位参与者特别提到，Copilot 提供的代码建议教会了他们之前不知道的东西。虽然可以用一维数组解决扫雷问题，但有些人可能会觉得使用二维数组更直观。不过，课程尚未讲授这部分内容。尽管如此，当 Copilot 自动生成使用二维数组的代码时，几位学生表示自己刚学到了新东西。另一个例子可以进一步说明这一点：

P08：“有一种情况是我没想到的，但它自动补全了代码，我当时就想：哦，看来我得考虑这种情况。”

几位参与者还表达了对在实践中使用 Copilot 以及其可能对学习产生负面影响的担忧。有些人担心自己无法学到在课堂上（进而在该领域）取得成功所必需的知识：

P03：“如果有人一直在用它，那他们实际上就没有在思考发生了什么，只是在按 Tab 键，也不清楚自己究竟在实现什么。” P06：“我不必学会编写代码，它会直接替我完成。” P08：“这会让我更不会解决问题，因为我会依赖它来帮我。”

学生们意识到，过度依赖 Copilot 生成的建议存在风险；这一观点在三分之一的访谈中出现。对于我们研究中的入门级问题，Copilot 大多数时候（但并非总是）能生成正确的解决方案 [34]。因此，当建议的代码有误时，过度依赖该工具尤其成问题，正如参与者 #11 所指出的：“我能想象自己会变得有点掉以轻心，到后来不再认真校对它建议的较长代码片段，从而漏掉一些问题。”学生们还担心，把 Copilot 当作拐杖会妨碍学习。参与者 #12 就说：“如果我用它，我会变得依赖它。上课时我可能会走神、不专心，因为我会想‘哦，这个我用 Copilot 就能做。’所以它会变成我的拐杖。”

4.3.2 提高速度。
学生认为，无需亲自输入代码以及获得解决问题的方法建议，都能提高效率：

P01：“如果我看到一个大致理解的提示词，可能会用它来自动填充一些小部分。” P02：“是的，这让我更快地想到接下来该怎么做。” P06：“如果不用它我也能编写程序，就不会用；如果我知道怎么解决，就会用它来提高速度。” P11：“要是让我自己输入整个循环，肯定要花很久；但它直接生成了代码，我只需要校对一下。”

一位参与者指出，将时间从输入代码转向思考问题，提升了效率：“快多了！不到20分钟我就开始测试了，而这段时间大部分都花在阅读问题和思考上。”Copilot 往往会生成语法正确的建议，因此可能特别有助于学生避免语法错误。访谈中，有几位参与者很好地说明了这一点：

P11：“另外，语法也是个大难题。它直接给出了语法，我只需要校对一下。” P17：“即便没有其他好处，它也能节省时间，省去反复检查该怎么做的过程。我喜欢让 Copilot 帮我处理语法，因为这一直是迄今为止我面临的最大挑战。”

尽管参与者对速度提升的评价大多是正面的，但并非所有人都认为避免输入代码以节省时间是一种好处。不过，我们研究中的一些参与者认识到，亲自输入代码可以作为一种练习，有助于学习；他们指出，接受Copilot建议可能会妨碍这一点：

P18：“我觉得自己敲代码有助于记住一些小事和细节。如果代码是别人直接给你的，你就会忘记一些小细节，比如分号，或者括号应该放在哪里。”

4.4 主题：推测

这个主题涵盖了所有关于 Copilot 未来可能的用途，以及可能出现的担忧或问题的陈述。该主题由三个子主题构成。这里的**智能** 指参与者认为 Copilot 具有一定程度智能的情况，例如它会“知道”某些事情。**未来** 指参与者对 Copilot 这类工具普及后世界面貌的推测，例如程序员因此失业。最后，如果参与者在访谈中的反思没有提到使用 Copilot 可能带来的任何负面影响，我们就将其标记为**没有缺点** 。

4.4.1 人工智能的自主性。
Copilot 对参与者来说是一种全新的体验，他们接触该工具的时间仅限于观察研究中使用的 30 分钟编程活动。在编码活动期间，几位参与者表示，他们觉得 Copilot 有意识、知识渊博，而且具有自主性。以下引文体现了这种感受：

> P13：“Copilot 知道我想做什么吗？感觉它生成的内容就像是我正在想的东西。” P19：“它在引导我找到答案。” P18：“它简直读懂了我的心思！” P15：“我觉得它知道我想要什么，这很奇怪。”

4.4.2 恐惧与担忧。
鉴于学生将智能归因于该系统，部分学生的回答带有推测性，并表达了对其未来可能被如何使用的担忧，这并不令人意外。例如，参与者#15担心：“它可能会接管世界，并编写自己的代码。”两名学生还担心 Copilot 可能会影响就业市场，或许会取代软件开发人员的工作。学生还担心 Copilot 可能引发涉及隐私的伦理问题，并不确定其对剽窃的影响。我们将在第5节进一步讨论这些问题。

5 讨论

现在，我们回到研究问题，讨论初学者首次使用 Copilot 等基于大语言模型的人工智能代码生成器时，我们的研究发现对新手程序员所产生的影响。随后，我们将讨论使用此类系统所引发的伦理考量。最后，我们将基于所呈现的全部研究发现与洞见，提出设计启示。

5.1 用户交互

我们的第一个研究问题是：“初学者第一次接触 GitHub Copilot 时，如何与其进行交互？”

由于 AI代码生成器 给出的代码建议可以被视为对学生当前程序的反馈，我们借助反馈理论视角讨论第一个研究问题的结果。我们依据 Hattie and Timperley 的反馈模型来考察 Copilot 的建议，该模型聚焦于三个反馈问题：“我要去哪里？”“进展如何？”以及“下一步该做什么？” [37]。

Copilot 主要为学生提供关于“下一步该做什么？”的反馈。我们发现初学者乐于使用 Copilot，并且既接受也拒绝了 Copilot 的代码建议。拒绝一些建议意味着至少有一些初学者认为 Copilot 的反馈可能是错误的。初学者使用 Copilot 既用于获取正确方向的初步指导，也用于在他们知道自己想要什么时创建代码。这些与 Barke 等人先前的工作一致，他们将有经验的程序员对 Copilot 的使用分为“探索”和“加速” [8]。从

反馈的角度来看，探索可以被视为学生试图从 Copilot 获取关于他们如何解决问题的初步想法的反馈（“我要去哪里？”），而加速则更符合获取关于当前实现策略的反馈（“我该如何进行？”）。

除了探索和加速之外，我们还观察到两种新颖的行为类型，分别称为“引导”和“漂移”。在引导行为中，学生大部分时间都在尝试促使 Copilot 生成代码；对于这组新手用户，我们认为这可能表明他们过度依赖工具。这与早期研究结果类似：学生有时会过度依赖自动化评估系统提供的自动生成的反馈 [7]。针对这类系统，先前研究提出了多种应对过度依赖的方法，例如设置提交惩罚 [7, 48]，或限制学生可以提交的次数 [39, 48]。一个值得探讨的问题是，是否也应对使用 Copilot 等人工智能代码生成器的初学者施加类似限制，以遏制过度依赖。从反馈的角度来看，如果只关注 Copilot 最擅长的“接下来该做什么？”，可能会使新手学生走上错误的解决方案路径——这与先前研究类似：Copilot 曾使有经验的程序员陷入“调试死胡同” [80]。

在我们观察到的另一种新颖行为“漂移”中，学生犹豫地接受 Copilot 的建议，可能还会尝试一番，但最终又回溯并删除代码，随后从头开始重复这一循环。从反馈的角度来看，学生可能会对人工智能系统的反馈心存怀疑——先前研究发现，人类对人工智能建议的信任取决于任务本身，以及人类在多大程度上认为人工智能有能力针对该任务提出建议 [81]。此外，现有的大多数编程自动化反馈系统更侧重于识别错误（“我做得怎么样？”和“我正朝哪个方向前进？”），而较少提供可执行的反馈 [43],，因此学生可能不习惯接收关于“接下来该怎么做？”的自动化反馈。

参与者似乎认为 Copilot 总能生成有用且正确的代码。对于初学者来说，这种情况可能尤为明显，因为他们可能熟悉自动化评估系统，例如，这类系统的反馈依据教师编写的单元测试 [43]，而在这种情况下，通常可以假定反馈是正确的。初学者往往认为编程情境中的反馈就是真相，并认为生成反馈的系统不会出错 [46]。然而，在使用 Copilot 等基于大语言模型的人工智能代码生成器时，这种看法会带来问题——近期关于编程学习的研究发现，Codex（为 Copilot 提供支持的大语言模型）的正确率约为 70%–80% [34, 75],，这意味着 Copilot 在约 20%–30% 的情况下会给出错误建议。这对新手程序员尤其成问题，因为他们可能难以判断建议是否正确。

人们可能会认为，既然参与者是在提示系统生成 C++ 代码，它就可能生成采用较新版本特性（例如 C++20）的高级功能。然而，Copilot 这类 Transformer 模型最好被理解为是在延续收到的提示词。因此，Codex 生成的代码，其复杂程度会与作为提示输入的代码相近；如果输入的代码存在问题，还可能引入更多程序错误 [22]。

随着学生逐渐熟悉这款工具，这类问题很可能会随着时间推移而减少，甚至只需与 Copilot 进行一次交互就能有所改善；但认真思考与系统的首次交互仍然很重要。我们将在阐述设计启示时进一步讨论这一点（第 5.4 节）。此外，这些交互模式与 Vaithilingam 等人对有经验的编程学生所做研究中的情况十分相似 [80]。他们报告称，学生往往花太多时间阅读 Copilot 提供的大段代码块，而在思考、阅读和调试之间不断切换上下文，导致任务表现变差。总体而言，对于我们研究中的新手程序员来说，情况似乎更糟：他们花更多时间理解代码，也更容易感到困惑并被引入歧途。这是因为，正如 Chen 等人所写，该模型并未与用户的意图保持一致 [22]。在这里，一种

可能的情况是，用户（学生）将 Copilot 用作反馈机制，期望反馈具有可操作性且质量良好；而 Copilot 最初的目的是充当用户的“AI 搭档程序员”，因此用户可能会预期某些建议并不值得深入探索。

初学者认为，使用 Copilot 的主要益处是加快了他们完成编程任务的进度，这与 Barke 等人 [8] 和 Vaithilingam 等人 [80]报告的有经验的程序员研究结果一致。与直接输入代码相比，使用 Copilot 建议的另一个相关益处是可以避免语法错误。计算机教育文献指出，语法错误及其产生的错误消息，对初学者以及正在学习新编程语言的学生都构成了重大挑战 [9, 27]。从本质上说，Copilot 可能让学生能够在更高的抽象层次上工作，将精力用于思考程序的语义，而非语法。

相反，参与者逐字输入代码建议的“慢速接受”现象，也可能有益于学生学习。这种打字练习对学习者具有教学益处；研究人员也探索过类似的打字练习，让学习者输入代码示例，以帮助新手学生练习低层次的实现技能 [35, 47]。尽管 Copilot 为这类练习提供了良好机会，但人们很容易直接接受 Copilot 的建议。

5.2 用户感知

我们的第二个研究问题是：“初学者如何看待自己首次使用 GitHub Copilot 的体验？”

对于编程入门，Copilot 可以帮助新手程序员更快地编写代码，并避免出现编程版的“写作障碍”。Copilot 还可以充当元认知支架，引导学生从更高层次思考他们正在解决的问题，例如规划代码结构，然后借助 Copilot 创建各个组件。因此，我们主要从元认知和自我调节的理论视角讨论与第二个研究问题相关的结果。

编程是一种复杂的认知活动 [64] ，往往涉及深入的元认知知识和行为[54]。一个兼具认知与元认知层面的编程活动示例，是使用具体输入进行代码追踪，这项活动受工作记忆限制 [24]。我们实验中的几位参与者在遇到困难时拿出记事本和铅笔，试图解决问题；这是反思性元认知行为（如自我调节）的明确例子 [52]。其他参与者则遇到了 Prather等人所描述的元认知困难，例如，虽然写了很多代码，却仍离可运行的解决方案很远，但因此产生了虚假的成就感[71]。几位参与者误解了问题提示词，不得不多次回头查看；Prather等人也观察到了这种情况 [70]。还有一些参与者把系统当作遇到困难时可以求助的同事；此前的研究记录过学生在学习小组中共同调节学习的类似行为 [69], , 这也契合了 Copilot 是“你的 AI 结对程序员”这一理念。尽管新手程序员的元认知技能正变得日益重要 [54, 68], , 但像 Copilot 这样的工具显然有机会从一开始就为这些行为提供支架并加以强化。我们将在下文的设计启示部分讨论这一点（第 5.4 节）。

在介绍 Codex 模型的论文中，Chen 等人概述了代码生成模型可能带来的一系列潜在风险 [22]。其中首要的风险是开发者过度依赖生成的输出；他们指出，这种情况可能对正在学习编写代码的新手程序员影响尤为明显。事实上，在我们的研究中，当参与者被问及对这项新技术的担忧和顾虑时，最常提到的正是这一点。参与者承认，过度依赖可能会妨碍自身学习；计算机教育工作者也表达过类似的担忧 [32]。从自我调节的角度来看，学生需要提升

自我调节技能，以便自我控制对 Copilot 等工具的使用，避免对其产生过度依赖——至少在学生可以自行决定是否使用这些工具时是如此。事实上，我们假设，对 Copilot 等工具的过度依赖可能会削弱新手的元认知编程技能和行为。

自然地，在实施这些认知和元认知行为时，我们可能会因此产生情绪唤起。情绪是打造可用性设计的关键组成部分 [2, 18, 56]。初学编程的学生体验到负面情绪时，可能会直接对其自我效能感造成负面影响 [45]。自我效能是一种相关的元认知信念，也是与学生在计算机科学领域的成功或失败相关性最高的直接衡量指标之一[73]。Copilot这类工具的设计可能激发积极情绪、给予鼓励，进而提升自我效能感，尤其是对计算领域中传统上代表性不足的少数群体而言；这一潜力不容低估。女性和代表性不足的少数群体分别仅占计算领域学士学位获得者的18%和11%，且往往身处所谓的“管道泄漏”现象之中 [20]。与多数群体相比，负面经历对代表性不足群体的影响往往更大，导致他们退出该专业 [57]。Copilot目前的交互方式实际上可能会给新手用户带来认知困难、元认知困难以及负面情绪，从而对其自我效能感产生相反的影响。我们认为，我们提出的设计建议（第5.4节）有助于缓解这些问题。

5.3 伦理考量

近期，强大的人工智能代码生成模型不断发展，由此引发了许多复杂的伦理问题。这些问题包括模型训练所用的数据、由此产生的法律和社会层面的担忧，以及迫在眉睫的学术不端问题。我们发现，即使参与者在研究中仅短暂接触了 Copilot，他们也提出了隐私和就业能力等一系列伦理问题，这颇有意思，也表明一些学生可能一开始会认为 Copilot 对自己构成威胁。我们认为，教育工作者了解这些担忧并帮助学生认识 Copilot 等工具的影响十分重要。正如我们在第 5.4 节所讨论的，我们提出的问题对设计人员也同样相关。

5.3.1 学术不端。学术不端是许多学科领域普遍存在的问题 [59, 76]。AI代码生成器的普及使这一问题对计算机教育工作者而言尤为复杂，因为它们既增加了发生学术不端的可能性，又降低了此类行为被发现的可能性。研究表明，Codex 在入门级编程题上的表现优于平均水平的学生，因此，对于可能考虑作弊的学生而言，它是一种有效的工具 [34]。与合同作弊或抄袭其他学生的作业等传统作弊方式相比，人工智能生成的解答无需与他人沟通，因此被抓到的风险更低 [84]。此外，人工智能生成的代码在结构上多种多样，也能规避标准的抄袭检测工具。Biderman 和 Raff 表明，AI代码生成器能够完成入门级编程作业，而不会触发 MOSS（一种先进的代码抄袭检测工具）的怀疑 [13]。

一项近期关于编程作业中剽窃问题的相关文献系统综述报告了学生为剽窃行为寻找合理化理由的常见方式 [3]。其中包括：学生认为，只要对他人生成的代码进行修改需要付出一定努力，使用这些代码就是可以接受的。这引发了一个问题：对于学术提交而言，由 Copilot 生成、随后经学生修改的代码，能否算作学生自己的代码。大多数开发环境都会为基本语法元素提供一些标准的代码补全工具。Copilot 将这种自动补全交互扩展为建议大段代码，其中有些代码我们发现学生会选择逐字符输入。在

这种情况下，学生能否声称该程序是自己创建的？这让人想起经典的“忒修斯之船”[7]哲学问题：如果我们要认定学生编写了这份代码提交，那么与从头开始由学生编写的代码相比，由工具生成的代码究竟达到什么程度才算过多，仍然是一个悬而未决的问题。我们预计，学术诚信问题将受到重大影响，同时也显然需要更新对剽窃的定义[26]。

5.3.2 代码复用与知识产权。 作为教育工作者，我们的职责之一是教导学生了解在课堂之外复用代码时应承担的专业责任。公开可用的代码，例如用于训练 Codex 模型的代码仓库，可能受各种许可证约束。特别是，开源软件包中的代码通常依据 GPL许可证发布，该许可证规定，任何衍生作品都必须依据相同或等效的许可证条款分发。然而，当代码由人工智能模型生成时，如何注明来源并不总是明确。近期出现了一场法律争议，原因是 Copilot 有时会逐字生成大段来自开源代码仓库的代码，却没有明确标示来源。这意味着开发者可能在不知情的情况下使用代码并违反许可证条款。2022年11月，有人提起集体诉讼，声称 Copilot 侵犯了依据开源许可证分享代码的创作者的权利。[8] 在教学生使用人工智能代码生成器时，教育工作者还应向学生讲解模型的训练方式，让他们认识到其中的法律和伦理影响。

5.3.3 有害偏见。 众所周知，训练数据集中的偏见会导致后续人工智能模型产生偏见 [12, 74]。最近 Meta AI 推出的 Galactica 语言模型就以相当引人注目的方式展现了这一点。该模型使用数百万篇学术论文、教科书和讲义进行训练。[9] 上线三天后，在遭到大量伦理批评之后，由于有证据表明它会生成逼真却错误的信息，以及包含潜在有害刻板印象的内容，该模型被下线。代码生成模型也无法免于这些问题：训练数据与最终用户的需求可能不匹配，也可能延续更严重且有害的社会偏见。Codex 的开发者指出，可以通过提示词使其“生成带有种族主义、贬损性及其他有害内容的代码注释” [22], ，并且它生成的代码结构可能反映有关性别、种族、情绪、阶级、姓名结构及其他特征的刻板印象 [22]。

就数据本身而言，代码生成模型主要基于公共代码仓库进行训练，这引发了一个问题：这些仓库中的内容是否适合正在学习编程的初学者。例如，公共仓库中发布的代码风格可能与教育材料不太匹配，或者可能使用令初学者感到困惑的高级功能。此外，已有研究表明，AI 生成的代码有时可能包含安全漏洞，从而误导学习者养成不良的编码习惯 [65]。

随着学生开始更广泛地采用 Copilot 和类似的代码生成工具，教育工作者教导学生了解其更广泛的社会和伦理影响将变得越来越重要。

5.4 设计启示

本节将根据观察与参与者访谈数据中归纳出的主题，探讨由此产生的设计启示。这些界面设计考量旨在更好地支持新手程序员或初次使用者。我们认为，这些考量对有经验的程序员和专家级程序员而言可能不那么重要。因此，用户

[7]https://en.wikipedia.org/wiki/Ship_忒修斯之_船 [8]https://githubcopilotlitigation.com/ [9] https://galactica.org/explore/

应能够选择希望从 Copilot 接收何种反馈，并随着编程技能的提升和积累，调整甚至隐藏这些反馈。

5.4.1 提示词控制。学生对 Copilot 表示不满时，通常是因为可用性问题。具体而言，学生不喜欢在不需要帮助时看到建议，因为这会拖慢他们的速度。这表明有必要为初学者设计新的交互体验。目前，Copilot 会实时生成建议，并在用户未发出请求的情况下显示出来。此外，Copilot 为所有用户提供相同的交互体验。我们可以利用丰富的上下文信息——例如正在解决的问题类型以及用户的背景知识——来调整代码建议的生成方式和时机。正在学习编程的初学者或许能从这样的方式中受益：先自行尝试解决问题，并在需要时请求帮助，而不是在不需要建议时还得忽略它们。

当 Copilot 给出大量代码时，学生通常要花费大量时间和精力来理解这些建议。被接受的较长建议之后往往会被修改，甚至彻底删除。参与者能够比起较长建议更快地判断较短建议是否有用；而被接受的较短建议则较少被修改或删除。这表明，采用优先选择较短方案或过滤较长方案的选择算法，或许能让 Copilot 的建议对学生（尤其是初学者）更有用。我们假设，冗长的自动生成代码建议可能会增加初学者的认知负荷 [78] ，并呼吁研究人员在未来的工作中对此展开探讨。

5.4.2 元认知支架。 如上所述，Copilot的用户界面可能会增强或损害新手程序员的元认知。先前的研究表明，专门针对新手程序员提供增强反馈的系统可以提高效能 [29, 72]和元认知行为 [70]。我们研究中的一些参与者通过“橡皮鸭调试法”使用 Copilot 来克服元认知困难，但这似乎是一种不常见的行为，可以通过界面本身得到更好的支持。如图 1a 和 1b 所示，Copilot 将其建议显示为单行或整块灰色文本。通过在建议代码正上方提供一个不显眼的 UI 元素，让用户循环浏览不同的代码建议，可以支架用户在问题解决过程中的进展。这本质上是让 Copilot 再次调用 Codex API，并且在输入足够高的变化（温度）时，下一个代码建议至少应有所不同。这可以鼓励学生更多地参与探索 和引导 交互模式。另一种支架形式可以是 Copilot 仅生成概述程序一般结构的注释，类似于子目标标签 [63]。

5.4.3 通过可解释人工智能实现更好的心智模型。 在观察环节中，参与者常常对 Copilot 的代码生成能力及其生成的代码感到困惑。同样，在访谈中，我们研究的参与者担心 Copilot 会生成他们无法理解的可运行代码。像 Copilot 这样的系统需要帮助用户理解它在做什么，这对新手程序员可能特别有效。可解释人工智能（XAI）是指人工智能系统及其所做决策应能被人们理解。尽管研究人员对 XAI 的研究已近 40 年 [23], ，但随着现代机器学习在我们日常生活中变得更加频繁，它的重要性正在增加 [50]。虽然该领域的大多数工作都集中在算法上，但人机交互研究人员呼吁开展更多研究，探讨它如何影响使用或受益于这些人工智能系统的人类 [1]。根据 Wang 等人的说法，人类的推理方式为 XAI 技术提供了信息 [82]。他们建议的 XAI 设计技术包括支持假设生成和支持正向（数据驱动）推理。这些对于像 Copilot 这样由大语言模型驱动的系统来说是理想的，因为用户处于编写代码的循环模式中，

阅读 Copilot 的自动生成的建议，并接受或拒绝这些建议。在这个循环中，用户正在构建 Copilot 如何工作的心智模型，这会影响他们接下来如何尝试使用 Copilot。由于向大语言模型创建精确提示词可能成为越来越重要的工具（就像如今的“谷歌搜索”一样），我们认为在向用户展示模型时，利用以用户为中心的 XAI 设计技术很重要。因此，我们建议像 Copilot 这样的系统应帮助用户稍微窥探一下黑箱，例如它使用什么作为输入、置信度值（或可视化），以及它对用户技能水平的估计。例如，非常近期的模型，特别是 OpenAI 的 ChatGPT10，已经开始呈现支持对话式对话的用户界面，因此非常适合向用户解释底层决策。

5.4.4 道德设计。当前围绕代码复用与许可的法律争议（见第 5.3.2 节），源于代码生成模型是在代码仓库上训练的，而这些代码可能受规定其使用方式的许可证约束。这些模型很容易生成与训练数据中部分代码完全相同的代码。如果善意的用户看到这类代码建议，却没有获得相应的许可证信息或源代码仓库链接，就可能造成问题。事实上，已有许多报告指出，用户会设计提示词引导 Copilot 从受许可约束的代码仓库中生成大段代码，却未作适当署名。开发者可能会在无意间创建包含许可证彼此不兼容的代码片段的项目。

人工智能代码生成器可以通过更明确地向用户提示生成的代码与现有源代码相匹配，或隐藏可能不符合用户自定义许可使用标准的建议，来解决这一问题。例如，早期版本的 GitHub Copilot 内置了一项过滤功能，能够检测代码建议是否与 GitHub 上的公共代码相匹配。用户可以选择启用此过滤功能；启用后，匹配或近似匹配的内容将不会作为建议显示。计划于 2023 年发布的 Copilot 版本将会在生成的代码与代码仓库中的代码建议相匹配时，附上源代码仓库的引用。[11] 尽管某些代码片段可能会出现在多个代码仓库中，代码生成工具仍可根据代码提交日期进行筛选，从而链接到权威来源。这些设计功能或许有助于教育工作者教导学生如何道德地复用代码，也能帮助学生在生成的代码并非原创时更好地注明来源。

5.5 局限性

本研究存在多项局限。首先，出于机构审查委员会（IRB）的考量，我们没有录制屏幕或音频。不过，我们认为，对学生行为的观察结合转录后的访谈，足以让我们从有助于深入了解的层面认识他们与用户界面的交互。其次，研究中的部分情境更像是在实验室中进行的体验，而非学生平时解决课堂编程作业的方式。我们尽可能减轻了这一影响（见第 3.2 节），但这可能影响了参与者的工作方式以及他们与工具的交互。第三，尽管目前已有多种代码生成工具，我们只研究了 Copilot，因为在收集数据时，它是唯一容易获取的此类工具。最后，本研究于 2022年4月下旬进行。Copilot 于 2022年3月29日发布，留给我们开展复杂、多渠道数据收集的时间并不充裕。我们迅速开展研究，以揭示这项技术在编程入门课程中的潜在影响。研究期间，这些体验对我们的学生而言都很新颖。到本文撰写时，即使我们没有开展本研究，我们的许多学生也很可能已经

[10]https://openai.com/blog/chatgpt/ [11]
https://github.blog/2022-11-01-preview-referencing-public-code-in-github-copilot/

通过其他途径接触过 Copilot。随着时间推移，新奇效应会逐渐消退。我们的结果反映了学生初次接触 Copilot 时的行为与想法。

5.6 未来工作

未来工作有许多值得探索的方向。例如，研究学生长期使用Copilot的情况，比如跟踪整个学期，以了解学生与Copilot的交互是否以及如何随时间推移、随着编程技能提升而发生变化。此外，未来的研究还应探究我们观察到的一些现象背后的原因。例如，始终显示代码建议是否会增加认知负荷，这或许可以解释学生为何感到沮丧，也能说明为何以往针对有经验的程序员开展的研究没有报告类似发现。总之，我们认为，深入研究如何在编程入门课堂中最有效地融入人工智能代码生成器至关重要。

6 结论

本研究首次探索了初学者在一门典型的编程入门（计算机科学导论课程）课程中，通过观察和访谈使用Copilot完成常见初学者编程任务的情况。我们发现，初学者难以理解和使用Copilot，对此类工具可能带来的影响心存顾虑，但也乐于在未来的开发工作中更全面地整合该工具。我们还观察到两种新颖的交互模式，探讨了研究结果的伦理影响，并提出了设计启示。这些发现对于将人工智能代码生成器融入编程入门课程体系，以及设计更易用的工具以帮助新手用户大规模解决编程任务中“卡住”的问题，都具有重要意义。

参考文献

[1] Ashraf Abdul、Jo Vermeulen、Danding Wang、Brian Y. Lim 和 Mohan Kankanhalli。2018年。《面向可解释、可问责和可理解系统的趋势与发展轨迹：人机交互研究议程》。收录于《*2018*年 *CHI* 计算系统中的人为因素会议论文集》（加拿大魁北克省蒙特利尔）（*CHI '18*）。计算机协会，纽约州纽约市，美国，1–18。https://doi.org/10.1145/3173574.3174156[2] Anshu Agarwal 和 Andrew Meyer。2009年。《超越可用性：将情绪反应作为用户体验不可或缺部分进行评估》。收录于《*CHI '09* 计算系统中的人为因素扩展摘要》。计算机协会，纽约州纽约市，美国，2919–2930。https://doi.org/10.1145/1520340.1520420[3] Ibrahim Albluwi。2019年。《编程评估中的剽窃：系统综述》。《*ACM*计算机教育学汇刊》20卷，1期，文章6（2019年12月），28页。https://doi.org/10.1145/3371156[4] Joe Michael Allen 和 Frank Vahid。2021年。《以简洁的图形表示学生每周完成多个小型程序所付出的努力》。收录于《第*52*届 *ACM*计算机科学教育技术研讨会论文集》（线上活动，美国）（*SIGCSE '21*）。计算机协会，纽约州纽约市，美国，349–354。https://doi.org/10.1145/3408877.3432551[5] Desai Ankur 和 Deo Atul。2022年。《推出 Amazon CodeWhisperer：由机器学习驱动的编码助手》。https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/[6] Jacob Austin、Augustus Odena、Maxwell Nye、Maarten Bosma、Henryk Michalewski、David Dohan、Ellen Jiang、Carrie Cai、Michael Terry、Quoc Le 和 Charles Sutton。2021年。《使用大语言模型进行程序合成》。*arXiv* 预印本 *arXiv:2108.07732*（2021年）。https://doi.org/10.48550/ARXIV.2108.07732[7] Elisa Baniassad、Lucas Zamprogno、Braxton Hall 和 Reid Holmes。2021年。《制止（自动评分器）疯狂：通过回归惩罚抑制对自动评分器的过度依赖》。收录于《第*52*届 *ACM*计算机科学教育技术研讨会论文集》（线上活动，美国）（*SIGCSE '21*）。计算机协会，纽约州纽约市，美国，1062–1068。https://doi.org/10.1145/3408877.3432430[8] Shraddha Barke、Michael B. James 和 Nadia Polikarpova。2022年。《有根据的 Copilot：程序员如何与代码生成模型互动》。*arXiv* 预印本 *arXiv:2206.15000*（2022年）。https://doi.org/10.48550/ARXIV.2206.15000[9] Brett A. Becker、Paul Denny、Raymond Pettit、Durell Bouchard、Dennis J. Bouvier、Brian Harrington、Amir Kamil、Amey Karkare、Chris McDonald、Peter-Michael Osera、Janice L. Pearce 和 James Prather。2019年。《编译器错误消息被认为无济于事：基于文本的编程错误消息研究概览》。收录于《计算机科学教育中的创新与技术工作组报告论文集》（苏格兰阿伯丁）（*ITiCSE-WGR '19*）。计算机协会，纽约州纽约市，美国，177–210。https://doi.org/10.1145/3344429.3372508

[10] Brett A. Becker、Cormac Murray、Tianyi Tao、Changheng Song、Robert McCartney 和 Kate Sanders。2018年。《先修复第一个，忽略其余错误：处理多条编译器错误消息》。收录于《第49届 ACM计算机科学教育技术研讨会论文集》（美国马里兰州巴尔的摩）（SIGCSE '18）。计算机协会，纽约州纽约市，美国，634–639。https://doi.org/10.1145/3159450.3159453[11] Brett A. Becker 和 Keith Quille。2019年。《SIGCSE 的计算机科学导论课程五十年：编程入门教育研究演进综述》。收录于《第50届 ACM计算机科学教育技术研讨会论文集》（美国明尼苏达州明尼阿波利斯）（SIGCSE '19）。计算机协会，纽约州纽约市，美国，338–344。https://doi.org/10.1145/3287324.3287432[12] Emily M Bender。2019年。《语言技术伦理风险的类型学：探讨透明文档可以在哪些方面发挥作用》。收录于《人工智能的未来：语言、伦理与技术研讨会》。[13] Stella Biderman 和 Edward Raff。2022年。《利用预训练语言模型欺骗 MOSS 检测》。收录于《第31届 ACM国际知识管理会议论文集》（美国佐治亚州亚特兰大）（CIKM '22）。计算机协会，纽约州纽约市，美国，2933–2943。https://doi.org/10.1145/3511808.3557079[14] Rishi Bommasani、Drew A. Hudson、Ehsan Adeli 等。2021年。《基础模型的机遇与风险》。arXiv 预印本 arXiv:2108.07258（2021年）。https://doi.org/10.48550/ARXIV.2108.07258[15] Virginia Braun 和 Victoria Clarke。2006年。《在心理学中运用主题分析》。定性心理学研究 3卷，2期（2006年），77–101。https://doi.org/10.1191/1478088706qp063oa[16] Virginia Braun 和 Victoria Clarke。2019年。《反思反思性主题分析》。运动、锻炼与健康定性研究 11卷，4期（2019年），589–597。https://doi.org/10.1080/2159676X.2019.1628806[17] Virginia Braun 和 Victoria Clarke。2022年。《主题分析的概念与设计思维》。定性心理学 9卷，1期（2022年），3–26。https://doi.org/10.1037/qup0000196[18] Scott Brave 和 Cliff Nass。2007年。《人机交互中的情绪》。收录于《人机交互手册》。CRC出版社，103–118。[19] Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等。2020年。《语言模型是少样本学习者》。神经信息处理系统进展 33卷（2020年），1877–1901。[20] Tracy Camp、W. Richards Adrion、Betsy Bizot、Susan Davidson、Mary Hall、Susanne Hambrusch、Ellen Walker 和 Stuart Zweben。2017年。《CS新世代：计算机科学的发展》。ACM Inroads 8卷，2期（2017年5月），44–50。https://doi.org/10.1145/3084362[21] Bei Chen、Fengji Zhang、Anh Nguyen、Daoguang Zan、Zeqi Lin、Jian-Guang Lou 和 Weizhu Chen。2022年。《CodeT：通过生成的测试生成代码》。arXiv 预印本 arXiv:2207.10397（2022年）。https://arxiv.org/abs/2207.10397[22] Mark Chen、Jerry Tworek、Heewoo Jun 等。2021年。《评估基于代码训练的大语言模型》。arXiv 预印本 arXiv:2107.03374（2021年）。https://doi.org/10.48550/ARXIV.2107.03374[23] William J Clancey。1983年。《基于规则的专家系统的认识论——一种解释框架》。人工智能 20卷，3期（1983年），215–251。[24] Will Crichton、Maneesh Agrawala 和 Pat Hanrahan。2021年。《工作记忆在程序跟踪中的作用》。收录于《2021年 CHI 计算系统中的人为因素会议论文集》（日本横滨）（CHI '21）。计算机协会，纽约州纽约市，美国，文章56，13页。https://doi.org/10.1145/3411764.3445257[25] Tyne Crow、Andrew Luxton-Reilly 和 Burkhard Wuensche。2018年。《面向编程教育的智能辅导系统：系统综述》。收录于《第20届澳大拉西亚计算教育会议论文集》（澳大利亚昆士兰州布里斯班）（ACE '18）。计算机协会，纽约州纽约市，美国，53–62。https://doi.org/10.1145/3160489.3160492[26] Nassim Dehouche。2021年。《海量生成式预训练 Transformer（GPT-3）时代的剽窃》。科学与环境政治伦理 21卷（2021年），17–23。[27] Paul Denny、Brett A. Becker、Nigel Bosch、James Prather、Brent Reeves 和 Jacqueline Whalley。2022年。《转向新编程语言期间初学者的反思》。收录于《第53届 ACM计算机科学教育技术研讨会论文集》第1卷（美国罗得岛州普罗维登斯）（SIGCSE 2022）。计算机协会，纽约州纽约市，美国，948–954。https://doi.org/10.1145/3478431.3499314[28] Paul Denny、Andrew Luxton-Reilly 和 Ewan Tempero。2012年。《并非所有语法错误都一样》。收录于《第17届 ACM创新与计算机科学教育技术年会论文集》（以色列海法）（ITiCSE '12）。计算机协会，纽约州纽约市，美国，75–80。https://doi.org/10.1145/2325296.2325318[29] Paul Denny、James Prather、Brett A. Becker、Catherine Mooney、John Homer、Zachary C Albrecht 和 Garrett B. Powell。2021年。《为初学者设计编程错误消息：可读性及其构成因素》。收录于《2021年 CHI 计算系统中的人为因素会议论文集》（日本横滨）（CHI '21）。计算机协会，纽约州纽约市，美国，文章55，15页。https://doi.org/10.1145/3411764.3445696[30] Iddo Drori、Sarah Zhang、Reece Shuttleworth、Leonard Tang、Albert Lu、Elizabeth Ke、Kevin Liu、Linda Chen、Sunny Tran、Newman Cheng 等。2022年。《神经网络通过程序合成和少样本学习，以人类水平求解、解释并生成大学数学题》。美国国家科学院院刊 119卷，32期（2022年），e2123433119。[31] Karl Anders Ericsson 和 Herbert Alexander Simon。1993年。《口语报告分析》（第1版）。麻省理工学院出版社，马萨诸塞州剑桥。[32] Neil A Ernst 和 Gabriele Bavota。2022年。《人工智能驱动的软件开发已然到来：你该担心吗？》。IEEE Software 39卷，2期（2022年），106–110。[33] Zhangyin Feng、Daya Guo、Duyu Tang、Nan Duan、Xiaocheng Feng、Ming Gong、Linjun Shou、Bing Qin、Ting Liu、Daxin Jiang 等。2020年。《CodeBERT：面向编程语言和自然语言的预训练模型》。arXiv 预印本 arXiv:2002.08155（2020年）。https://doi.org/10.48550/arXiv.2002.08155[34] James Finnie-Ansley、Paul Denny、Brett A. Becker、Andrew Luxton-Reilly 和 James Prather。2022年。《机器人来了：探索 OpenAI Codex 对编程入门的影响》。收录于澳大拉西亚计算教育会议（线上活动，澳大利亚）（ACE '22）。

计算机协会，纽约州纽约市，美国，10–19。https://doi.org/10.1145/3511861.3511863[35] Adam M. Gaweda、Collin F. Lynch、Nathan Seamon、Gabriel Silva de Oliveira 和 Alay Deliwa。2020年。《将打字练习作为计算机科学课程中刻意练习的交互式示例演练》。载于《第二十二届澳大拉西亚计算教育会议论文集》（澳大利亚维多利亚州墨尔本）（ACE'20）。计算机协会，纽约州纽约市，美国，105–113。https://doi.org/10.1145/3373165.3373177[36] Lisa M Given。2015年。《关于定性研究的100个问题（及答案）》。SAGE publications。[37] John Hattie 和 Helen Timperley。2007年。《反馈的力量》。《教育研究评论》77，1（2007年），81–112。[38] Arto Hellas、Juho Leinonen 和 Petri Ihantola。2017年。《居家考试中的剽窃：寻求帮助、协作与有组织的作弊》。载于《2017年ACM计算机科学教育创新与技术会议论文集》（意大利博洛尼亚）（ITiCSE '17）。计算机协会，纽约州纽约市，美国，238–243。https://doi.org/10.1145/3059009.3059065[39] Michael S. Irwin 和 Stephen H. Edwards。2019年。《能否运用移动游戏心理学改善编程作业的时间管理？》。载于《ACM全球计算教育会议论文集》（中国四川成都）（CompEd '19）。计算机协会，纽约州纽约市，美国，208–214。https://doi.org/10.1145/3300115.3309517[40] Dhanya Jayagopal、Justin Lubin 和 Sarah E. Chasins。2022年。《探索新手程序员对程序合成器的可学习性》。载于《第35届ACM用户界面软件与技术年度研讨会论文集》（美国俄勒冈州本德）（UIST '22）。计算机协会，纽约州纽约市，美国，论文编号64，共15页。https://doi.org/10.1145/3526113.3545659[41] Ioannis Karvelas、Annie Li 和 Brett A. Becker。2020年。《编译机制与错误消息呈现方式对新手程序员行为的影响》。载于《第51届ACM计算机科学教育技术研讨会论文集》（美国俄勒冈州波特兰）（SIGCSE '20）。计算机协会，纽约州纽约市，美国，759–765。https://doi.org/10.1145/3328778.3366882[42] Ayaan M. Kazerouni、Riffat Sabbir Mansur、Stephen H. Edwards 和 Clifford A. Shaffer。2019年。《学生的调试实践及其与项目成果的关系》。载于《第50届ACM计算机科学教育技术研讨会论文集》（美国明尼苏达州明尼阿波利斯）（SIGCSE '19）。计算机协会，纽约州纽约市，美国，1263。https://doi.org/10.1145/3287324.3293794[43] Hieke Keuning、Johan Jeuring 和 Bastiaan Heeren。2018年。《编程练习自动反馈生成的系统性文献综述》。《ACM计算教育汇刊》19，1，论文编号3（2018年9月），共43页。https://doi.org/10.1145/3231711[44] Päivi Kinnunen 和 Beth Simon。2010年。《体验计算机科学导论课程中的编程作业：情绪代价》。载于《第六届计算教育研究国际研讨会论文集》（丹麦奥胡斯）（ICER '10）。计算机协会，纽约州纽约市，美国，77–86。https://doi.org/10.1145/1839594.1839609[45] Päivi Kinnunen 和 Beth Simon。2011年。《计算机科学专业学生对计算机科学导论课程中自我效能的认知：从社会认知理论视角得出的结果》。载于《第七届计算教育研究国际研讨会论文集》（美国罗得岛州普罗维登斯）（ICER '11）。计算机协会，纽约州纽约市，美国，19–26。https://doi.org/10.1145/2016911.2016917[46] Michael J. Lee 和 Amy J. Ko。2011年。《将编程工具反馈拟人化有助于新手程序员学习》。载于《第七届计算教育研究国际研讨会论文集》（美国罗得岛州普罗维登斯）（ICER '11）。计算机协会，纽约州纽约市，美国，109–116。https://doi.org/10.1145/2016911.2016934[47] Antti Leinonen、Henrik Nygren、Nea Pirttinen、Arto Hellas 和 Juho Leinonen。2019年。《探索简单语法书写练习对编程学习的适用性》。载于《第50届ACM计算机科学教育技术研讨会论文集》（美国明尼苏达州明尼阿波利斯）（SIGCSE '19）。计算机协会，纽约州纽约市，美国，84–90。https://doi.org/10.1145/3287324.3287378[48] Juho Leinonen、Paul Denny 和 Jacqueline Whalley。2022年。《比较编程入门项目中的即时反馈与定时反馈》。载于《第53届ACM计算机科学教育技术研讨会论文集》第1卷（美国罗得岛州普罗维登斯）（SIGCSE 2022）。计算机协会，纽约州纽约市，美国，885–891。https://doi.org/10.1145/3478431.3499372[49] Yujia Li、David Choi、Junyoung Chung 等。2022年。《利用AlphaCode实现竞赛级代码生成》。arXiv预印本 arXiv:2203.07814（2022年）。https://doi.org/10.48550/ARXIV.2203.07814[50] Q. Vera Liao、Daniel Gruen 和 Sarah Miller。2020年。《质疑人工智能：为可解释人工智能用户体验提供设计实践依据》。载于《2020年CHI人机因素计算系统会议论文集》（美国夏威夷州檀香山）（CHI '20）。计算机协会，纽约州纽约市，美国，1–15。https://doi.org/10.1145/3313831.3376590[51] Pengfei Liu、Weizhe Yuan、Jinlan Fu、Zhengbao Jiang、Hiroaki Hayashi 和 Graham Neubig。2022年。《预训练、提示与预测：自然语言处理提示方法的系统性综述》。《ACM计算调查》（2022年9月）。https://doi.org/10.1145/3560815[52] Dastyni Loksa 和 Amy J. Ko。2016年。《自我调节在编程问题解决过程及成功中的作用》。载于《2016年ACM国际计算机教育研究会议论文集》（澳大利亚维多利亚州墨尔本）（ICER '16）。计算机协会，纽约州纽约市，美国，83–91。https://doi.org/10.1145/2960310.2960334[53] Dastyni Loksa、Amy J. Ko、Will Jernigan、Alannah Oleson、Christopher J. Mendez 和 Margaret M. Burnett。2016年。《编程、问题解决与自我觉察：明确指导的影响》。载于《2016年CHI人机因素计算系统会议论文集》（美国加利福尼亚州圣何塞）（CHI '16）。计算机协会，纽约州纽约市，美国，1449–1461。https://doi.org/10.1145/2858036.2858252[54] Dastyni Loksa、Lauren Margulieux、Brett A. Becker、Michelle Craig、Paul Denny、Raymond Pettit 和 James Prather。2022年。《编程教育中的元认知与自我调节：理论及应用范例》。《ACM计算教育汇刊》22，4，论文编号39（2022年9月），共31页。https://doi.org/10.1145/3487050[55] Mariam Mahdaoui、Said Nouh、My Seddiq Elkasmi Alaoui 和 Mounir Sadiq。2022年。《智能编程辅导系统中自动提示生成方法的比较研究》。《Procedia计算机科学》198（2022年），391–396。https://doi.org/10.1016/j.procs.2021.12.259 第12届

新兴泛在系统与普适网络国际会议 / 信息与通信技术在医疗保健领域的当前与未来趋势第11届国际会议。[56] Lauri Malmi、Judy Sheard、Päivi Kinnunen、Simon 和 Jane Sinclair。2020年。《编程教育背景下的情绪、态度与自我效能理论和模型》。载于《2020年ACM国际计算机教育研究会议论文集》（线上会议，新西兰）（ICER '20）。计算机协会，纽约州纽约市，美国，36–47。https://doi.org/10.1145/3372782.3406279[57] Jane Margolis 和 Allan Fisher。2002年。《走进俱乐部会所：计算领域的女性》。麻省理工学院出版社。[58] Jessica McBroom、Irena Koprinska 和 Kalina Yacef。2021年。《自动生成编程提示综述：HINTS框架》。《ACM计算调查》54，8，论文编号172（2021年10月），共27页。https://doi.org/10.1145/3469885[59] Donald L. McCabe、Linda Klebe Trevino 和 Kenneth D. Butterfield。2001年。《学术机构中的作弊：十年研究》。《伦理与行为》11，3（2001年），219–232。https://doi.org/10.1207/S15327019EB1103_2[60] Robert McCartney、Anna Eckerdal、Jan Erik Mostrom、Kate Sanders 和 Carol Zander。2007年。《成功学生摆脱困境的策略》。载于《第12届SIGCSE计算机科学教育创新与技术会议论文集》（苏格兰邓迪）（ITiCSE '07）。计算机协会，纽约州纽约市，美国，156–160。https://doi.org/10.1145/1268784.1268831[61] Nora McDonald、Sarita Schoenebeck 和 Andrea Forte。2019年。《定性研究中的可靠性与评定者间信度：计算机支持的协同工作与人机交互实践的规范和指南》。《ACM人机交互论文集》3，CSCW，论文编号72（2019年11月），共23页。https://doi.org/10.1145/3359174[62] Charlie McDowell、Linda Werner、Heather Bullock 和 Julian Fernald。2002年。《结对编程对编程入门课程表现的影响》。《SIGCSE简报》34，1（2002年2月），38–42。https://doi.org/10.1145/563517.563353[63] Briana B. Morrison、Lauren E. Margulieux 和 Mark Guzdial。2015年。《计算问题求解学习中的子目标、背景与示例演练》。载于《第十一届国际计算机教育研究年度会议论文集》（美国内布拉斯加州奥马哈）（ICER '15）。计算机协会，纽约州纽约市，美国，21–29。https://doi.org/10.1145/2787622.2787733[64] Tom Ormerod。1990年。《人类认知与编程》。载于《编程心理学》。爱思唯尔，63–82。[65] Hammond Pearce、Baleegh Ahmad、Benjamin Tan、Brendan Dolan-Gavitt 和 Ramesh Karri。2022年。《键盘前睡着了吗？评估GitHub Copilot的代码贡献安全性》。载于《2022年IEEE安全与隐私研讨会》（SP），754–768。https://doi.org/10.1109/SP46214.2022.9833571[66] H. Pearce、B. Tan、B. Ahmad、R. Karri 和 B. Dolan-Gavitt。2023年。《考察使用大语言模型进行零样本漏洞修复》。载于《2023年IEEE安全与隐私研讨会》（SP）。IEEE计算机学会，洛斯阿拉米托斯，美国，1–18。https://doi.org/10.1109/SP46215.2023.00001[67] Hammond Pearce、Benjamin Tan、Prashanth Krishnamurthy、Farshad Khorrami、Ramesh Karri 和 Brendan Dolan-Gavitt。2022年。《突击测验！大语言模型能否助力逆向工程？》arXiv预印本 arXiv:2202.01142（2022年）。https://doi.org/doi.org/10.48550/arXiv.2202.01142[68] James Prather、Brett A. Becker、Michelle Craig、Paul Denny、Dastyni Loksa 和 Lauren Margulieux。2020年。《我们认为自己在做什么？编程中的元认知与自我调节》。载于《2020年ACM国际计算机教育研究会议论文集》（线上会议，新西兰）（ICER '20）。计算机协会，纽约州纽约市，美国，2–13。https://doi.org/10.1145/3372782.3406263[69] James Prather、Lauren Margulieux、Jacqueline Whalley、Paul Denny、Brent N. Reeves、Brett A. Becker、Paramvir Singh、Garrett Powell 和 Nigel Bosch。2022年。《在朋友的帮助下应对：将编程入门课程中的小组学习理解为社会共享调节》。载于《2022年ACM国际计算机教育研究会议论文集》第1卷（瑞士卢加诺及线上会议）（ICER '22）。计算机协会，纽约州纽约市，美国，164–176。https://doi.org/10.1145/3501385.3543970[70] James Prather、Raymond Pettit、Brett A. Becker、Paul Denny、Dastyni Loksa、Alani Peters、Zachary Albrecht 和 Krista Masci。2019年。《先做最重要的事：为理解问题提示提供元认知支架》。载于《第50届ACM计算机科学教育技术研讨会论文集》（美国明尼苏达州明尼阿波利斯）（SIGCSE '19）。计算机协会，纽约州纽约市，美国，531–537。https: //doi.org/10.1145/3287324.3287374[71] James Prather、Raymond Pettit、Kayla McMurry、Alani Peters、John Homer 和 Maxine Cohen。2018年。《新手程序员在自动评测工具中面临的元认知困难》。载于《2018年ACM国际计算机教育研究会议论文集》（芬兰埃斯波）（ICER '18）。计算机协会，纽约州纽约市，美国，41–50。https://doi.org/10.1145/3230977.3230981[72] James Prather、Raymond Pettit、Kayla Holcomb McMurry、Alani Peters、John Homer、NevanSimone 和 Maxine Cohen。2017年。《新手与编译器错误消息的交互：人为因素视角》。载于《2017年ACM国际计算机教育研究会议论文集》（美国华盛顿州塔科马）（ICER '17）。计算机协会，纽约州纽约市，美国，74–82。https://doi.org/10.1145/3105726.3106169[73] Ashley D Rittmayer 和 Margaret E Beier。2008年。《概述：科学、技术、工程和数学领域中的自我效能》。SWE-AWE CASEE综述 1，3（2008年），12。[74] Drew Roselli、Jeanna Matthews 和 Nisha Talagala。2019年。《人工智能中的偏见管理》。载于《2019年万维网会议伴随论文集》（美国旧金山）（WWW '19）。计算机协会，纽约州纽约市，美国，539–544。https://doi.org/10.1145/3308560.3317590[75] Sami Sarsa、Paul Denny、Arto Hellas 和 Juho Leinonen。2022年。《使用大语言模型自动生成编程练习与代码解释》。载于《2022年ACM国际计算机教育研究会议论文集》第1卷（瑞士卢加诺及线上会议）（ICER '22）。计算机协会，纽约州纽约市，美国，27–43。https://doi.org/10.1145/3501385.3543957[76] Simon、Judy Sheard、Michael Morgan、Andrew Petersen、Amber Settle、Jane Sinclair、Gerry Cross 和 Charles Riedesel。2016年。《在计算教育中破解学术诚信迷局》。载于《2016年ITiCSE工作组报告论文集》（秘鲁阿雷基帕）（ITiCSE '16）。计算机协会，纽约州纽约市，美国，57–80。https://doi.org/10.1145/3024906.3024910[77] E. Soloway。1986年。《学习编程 = 学习构建机制与解释》。《ACM通讯》29，9（1986年9月），850–858。https://doi.org/10.1145/6592.6594[78] John Sweller。1994年。《认知负荷理论、学习困难与教学设计》。《学习与教学》4，4（1994年），295–312。

[79] Leonard Tang、Elizabeth Ke、Nikhil Singh、Bo Feng、Derek Austin、Nakul Verma 和 Iddo Drori。2022年。通过概率程序合成人类水平地解决概率与统计问题并预测问题的可解性。载于国际人工智能教育会议。Springer，612–615。[80] Priyan Vaithilingam、Tianyi Zhang 和 Elena L. Glassman。2022年。期望与体验：评估由大语言模型驱动的代码生成工具的可用性。载于*CHI*计算机系统人因会议扩展摘要。计算机协会，纽约，纽约州，美国，1–7。[81] KailasVodrahalli、RoxanaDaneshjou、TobiasGerstenberg 和 JamesZou。2022年。人们是否更信任来自人工智能的建议？人机交互分析。载于*2022*年*AAAI/ACM*人工智能、伦理与社会会议论文集（英国牛津）*(AIES '22)*。计算机协会，纽约，纽约州，美国，763–777。https://doi.org/10.1145/3514094.3534150[82] Danding Wang、Qian Yang、Ashraf Abdul 和 Brian Y. Lim。2019年。设计理论驱动、以用户为中心的可解释人工智能。载于*2019*年*CHI*计算机系统人因会议论文集（苏格兰格拉斯哥）*(CHI '19)*。计算机协会，纽约，纽约州，美国，1–15。https://doi.org/10.1145/3290605.3300831[83] Jason Wei、Yi Tay、Rishi Bommasani、Colin Raffel、Barret Zoph、Sebastian Borgeaud、Dani Yogatama、Maarten Bosma、Denny Zhou、Donald Metzler 等人。2022年。大语言模型的涌现能力。*arXiv*预印本 *arXiv:2206.07682*（2022年）。https://doi.org/10.48550/arXiv.2206.07682[84] Jonathan Yorke、Lesley Sefcik 和 Terisha Veeran-Colton。2022年。合同作弊与敲诈：一桩危险的交易？《高等教育研究》47，1（2022年），53–66。https://doi.org/10.1080/03075079.2020.1730313[85] Sarah Zhang、Reece Shuttleworth、Derek Austin、Yann Hicke、Leonard Tang、Sathwik Karnik、Darnell Granberry 和 Iddo Drori。2022年。用于自动解答和生成机器学习期末考试题的数据集与基准测试。*arXiv*预印本 *arXiv:2206.05442*（2022年）。https://doi.org/10.48550/ARXIV.2206.05442