Vibe-Check 协议：量化人工智能编程中的认知卸载

Aizierjiang Aiersilan 乔治·
华盛顿大学
alexandera@gwu.edu

arXiv:2601.02410v1 [cs.SE] 2 Jan 2026

摘要

将大语言模型（LLMs）融入软件工程教育，推动了“氛围编程”的出现。这一范式要求开发者通过自然语言阐述高层次意图，并将实现工作委托给人工智能代理。尽管支持者认为，这种方法通过强调概念设计而非语法记忆，使教学法实现了现代化，但不断积累的实证证据引发了人们对技能保持和深层概念理解的担忧。本文提出一个理论框架，用于探讨以下研究问题：氛围编程是否是学习软件工程的更好方式？我们认为，在利用人工智能加速工作与利用人工智能进行认知卸载的学生之间，学生成果存在差异。为评估这些教育权衡，我们提出**Vibe-Check 协议（VCP）**，这是一种系统性的基准测试框架，包含三个定量指标：用于建模技能衰退的冷启动重构（M_{CSR}）；基于信号检测理论、用于评估错误识别能力的幻觉陷阱检测（M_{HT}）；以及用于量化代码复杂度与概念理解之间差异的可解释性差距（E_{gap}）。通过受控比较，VCP 旨在为教育工作者确定最佳教学边界提供定量依据：识别氛围编程能够促进真正掌握的情境，以及它会引入隐性技术债务和表面能力的情境。

1 引言

在快速发展的计算机科学教育领域，大语言模型（LLM）的出现，显著改变了学生参与编程任务和概念学习的方式。这种转变最明显地体现在一种工作流程的广泛采用上，Andrej Karpathy 将其称为“氛围编程”：这是一种教学与实践方法，程序员不再直接控制代码生成的语法细节，而是通过自然语言交互阐述高层次意图，引导 Cursor、Claude Code 和 OpenHands 等先进的人工智能代理 [Karpathy, 2025, Wang 等人， 2024]。这一转变不仅仅是采用了一种新工具；它改变了程序员与代码之间的关系，可能使编程从直接操控转变为人类意图与机器实现之间的协作对话。

这一新兴现象引发了一项关键的教学探究：氛围编程是否是学习软件工程原理与实践的更优方法，还是仅仅

营造出一种精通的逼真假象，掩盖了理解上的根本性缺口？这两种可能性之间的区别虽微妙，却对教育政策、课程设计和长期学生成果有着深远影响。实证观察显示，学生学习轨迹存在复杂的分化：一些学习者将这些先进工具用作真正的“力量倍增器”，加速复杂架构模式的实现。然而，令人担忧的是，相当一部分学生似乎陷入了严重“认知卸载”的模式：他们能够成功开发出可运行的应用程序，但在移除人工智能辅助后，却无法令人信服地解释、修改或扩展其底层逻辑。

学术界对这一分歧的回应各不相同。斯坦福大学等进步院校通过 CS146S：现代软件开发者等创新课程，拥抱了这一技术理念；该课程认为，传统的语法记忆已是一种过时的能力 [Eric, 2025]。然而，业界讨论显示，人们日益担忧随意的“氛围编程”与严谨的“AI 辅助工程”之间的区别：前者以快速、凭直觉的提示为特征，缺乏深入审查；后者则坚持严格的人工监督 [Osmani, 2025]。Zhao 等人近期开展的基准测试 [2025]显示，尽管人工智能代理的功能正确率很高，但相当一部分解决方案仍含有潜在缺陷和安全漏洞，这引发了人们对完全由 AI 驱动的代码生成流程能否长期可行的疑问。

从教育角度来看，风险在于“能力错觉” [Rojas-Galeano,2025]。这种错觉根植于类似邓宁-克鲁格效应的元认知偏差 [Kruger 和 Dunning, 1999], ：学生会将生成代码的能力误认为理解代码的能力。为应对这一问题，本文提出了 **Vibe-Check 协议（VCP）**，这是一种旨在量化氛围编程教育影响的新型基准测试框架。与以往的定性研究不同，VCP 采用系统的数学建模，衡量技能保持、错误检测敏感度和概念理解。

本研究的贡献有两方面。首先，它为教育工作者提供了一套形式化的指标集，用于将氛围编程与传统学习方法进行基准比较。其次，它提出了一个用于课程设计的决策理论框架，帮助教育工作者辨别哪些具体教学情境适合借助 AI 加速学习，哪些情境则仍需通过传统的语法探索来实现深度学习。

本文其余部分安排如下：第 2 节回顾相关文献。第 3 节详细介绍所提出的 VCP方法论，并给出技能保持与理解的精确数学表述。第 4 节探讨该框架的理论启示与期望效用。第 5 节说明研究局限，第 6 节总结全文。

2 文献综述与问题陈述

2.1 氛围编程的兴起及其制度化应用

氛围编程的兴起，是对人工智能工具在软件开发情境中日益普及且不断成熟的直接回应，代表着对程序员在代码创建过程中的角色进行根本性重新构想。Karpathy 的

奠基性概念将这种方法描述为一种本质上以对话为基础的工作流程：用户用自然语言表达意图、需求和限制，“顺应氛围”，从而让大语言模型处理复杂的实现细节、语法要求和底层优化决策，而这些工作过去通常会耗费开发者大量时间和认知资源 [Karpathy, 2025]。这一范式与人机协作在多个领域的广泛趋势高度契合；GitHub Copilot、Cursor 和 Claude Code 等工具通过先进的提示工程实现快速原型设计和迭代开发，而非采用需要直接操作语法和控制结构的传统逐行编码方式。

从历史角度来看，氛围编程建立在早期几代人工智能辅助编程工具的基础之上，包括代码补全系统、智能语法高亮工具和自动重构工具，但它在程序员的主要角色以及其与代码进行认知互动的方式上，带来了性质不同且更为根本的结构性变化。以 GitHub Copilot 等以往工具为例，它们主要根据即时上下文提供单个函数的建议或补全部分实现；而氛围编程则通过迭代对话，指挥 AI 智能体构建完整模块、实现整套功能，甚至设计系统级解决方案，并逐步完善需求与实现方案。这一区别从根本上减少了直接操作代码和掌握语法专业知识的必要性，使编程从一项以技术为主的技能，转变为更具协作性和沟通性的过程，强调清晰表达意图，而非亲自执行实现。

业内从业者对这些方法之间的区别进行了细致记录，并将相对不加批判地接受 AI 建议、缺乏深入技术审查的随意“氛围编程”，与成熟的“人工智能辅助工程”区分开来；在后者中，工程师通过结构化设计文档、全面的代码审查流程和严格的测试驱动开发方法，保持系统性的掌控 [Osmani, 2025]。这种专业层面的区分凸显了在 AI 增强型开发环境中保持人工监督和技术判断的重要性，也表明要有效应用这些技术，需要扎实的领域专业知识，而非取代深厚编程知识的必要性。

机构对氛围编程方法的采用，最突出的例子或许是斯坦福大学开设的创新课程 CS146S。该课程明确将学生定位为构建现代软件系统时 AI 智能体的协调者和指挥者，而非专注于掌握语法和实现算法的传统实现者 [Eric,2025]。课程大纲强调高级提示词设计技巧、系统化设计思维、迭代优化流程以及人机协作工作流，并有力主张：与传统的死记硬背式语法记忆或底层调试能力相比，这些技能更符合当代行业需求，也更具价值。这种方法的支持者认为，氛围编程大幅降低了复杂编程任务的入门门槛，使学生能够将认知资源集中于高阶思维、系统架构和创造性

问题解决，而不是陷入实现细枝末节。早期采用者的轶事报告显示，学生已能在数周或数月内成功开发出复杂的全栈应用程序、分布式系统和创新的软件解决方案；按照传统方式，这些项目可能需要整整一学期的常规教学与练习。然而，这种显著提升的项目完成效率，可能绕过了刻意练习和富有成效的努力，而专长研究一再指出，这些对于真正实现领域精通至关重要，因此也引发了关于此类加速学习方式长期教育影响的根本性问题 [Ericsson 等人,1993]。

2.2 认知风险与能力错觉

尽管效率有所提升，实证证据仍揭示了显著的教学风险。“认知卸载”（即依赖外部辅助会削弱内在加工）这一概念在认知心理学中已有充分记载 [Risko 和 Gilbert， 2016]。在教育情境中，这表现为“浅层学习”：学习者虽然达到表面熟练，却缺乏真正专业能力所具有的概念深度 [Marton 和 S¨alj¨o， 1976]。

Lau 和 Guo 关于教师适应策略的研究指出，学生可能会绕过概念理解 [Lau 和 Guo， 2023]。Rojas-Galeano 的实证课堂研究也证实了这一发现。该研究采用两阶段方案：20 名本科生先在人工智能辅助下完成编程任务，随后在没有支持的情况下尝试扩展任务 [Rojas-Galeano， 2025]。结果显示出明显的脱节：在人工智能辅助阶段认为自己理解程度较高的学生，在独立完成任务时表现显著下降。定量分析发现，学生自我报告的无人工智能辅助工作能力与其对无辅助任务难度的感知之间存在中等至较强的相关性（$\rho = 0.58$， $p = .008$），这表明遇到最大困难的学生也意识到了自己的依赖。定性分析归纳出一些常见主题，包括“人工智能掩盖了不完整的理解”（$n = 3$）和“难以迁移知识”。一名学生指出：“你以为自己理解了代码的运行方式，但试着独立创建代码时，就会发现自己其实没有完全弄懂。”

“能力错觉”是这一批判的核心。它根植于邓宁-克鲁格效应：能力较低的人由于缺乏元认知意识而高估自己的能力 [Kruger 和 Dunning， 1999], 当学生将功能性代码误认为掌握了相关知识时，这种错觉便会显现。Rojas-Galeano 的数据表明，学生在人工智能辅助任务中报告了中等至较高的信心，但在独立完成任务时，他们的表现显著下滑，许多人甚至难以完成自己据称编写的代码的简单扩展[Rojas-Galeano， 2025]。

2.3 人工智能生成代码的质量与信度问题

氛围编程的影响不仅限于教育方面，也涉及生产软件质量。Zhao 等人的 SUSVIBES 基准测试对智能体生成的代码进行了全面评估 [Zhao 等人， 2025]。他们的研究结果表明，尽管 SWE-Agent 等智能体[Yang 等人， 2024] 能够实现较高的功能正确性，但相当一部分解决方案仍包含

潜在缺陷或漏洞。这表明，仅靠功能测试不足以保证代码质量。

Pearce 等人对 GitHub Copilot 贡献的分析印证了这些发现，表明人工智能生成的代码可能引入开发者未能察觉的细微程序错误 [Pearce et al., 2025]。行业调查报告指出，尽管人工智能能够加快开发速度，却可能造成“信任债务”，即累积起来的、虽能运行但无人理解的代码所带来的负担 [Osmani,2025]。正如一位首席技术官所观察到的，人工智能生成的代码“看似运行完美，直到灾难性故障突然发生”，这凸显了深入理解对于长期维护的必要性。

2.4 当前评估框架中的不足

现有的人工智能生成代码及其教育影响评估方法存在显著局限。基准测试往往侧重于代码质量，却完全忽视教育成果。Lau 和 Guo 以及 Rojas-Galeano 等人的教学研究提供了宝贵的定性见解，但缺乏可用于课堂评估的定量、可扩展指标。

当前领域存在若干具体不足。首先，缺乏纵向设计：大多数研究只捕捉单次课程的情况，而非长期追踪技能发展。其次，人工智能辅助与无辅助表现之间的比较很少控制任务复杂度，因此缺乏难度标准化。第三，教育指标往往未经系统设计便临时制定，既未确立信度，也未确立效度，导致心理测量学验证有限。第四，研究虽能发现问题，却很少提出或评估纠正策略，表明干预框架仍然缺失。据我们所知，尚无框架能够弥合代码质量评估与教育成果测量之间的差距。

VCP 通过提供一种结构化协议来弥补这些空白，该协议以整体方式衡量学习情况，纳入纵向追踪，控制混杂变量，并提出循证干预措施。通过将错误检测指标（通过幻觉陷阱检测）与技能保持和理解度量相结合，VCP 旨在为教育工作者提供在人工智能辅助学习时代进行课程设计所需的可操作洞见。

3 方法论：Vibe-Check 协议（VCP）

Vibe-Check 协议（VCP）是一个旨在通过受控的纵向实验评估氛围编程教育影响的框架。该协议借鉴了教育测量领域的成熟实践以及近期基准测试的见解 [Zhao 等人， 2025], 包含三项主要指标，分别针对学习的不同维度：保持、错误检测和理解（见图 1）。该协议将学生与接受传统教学的学生进行基线比较，通过标准化评分细则控制任务难度，并采用试点测试和评分者间信度评估进行心理测量学验证。伦理考量至关重要，所有参与者均须提供知情同意，并接受关于实验流程的完整实验后说明。

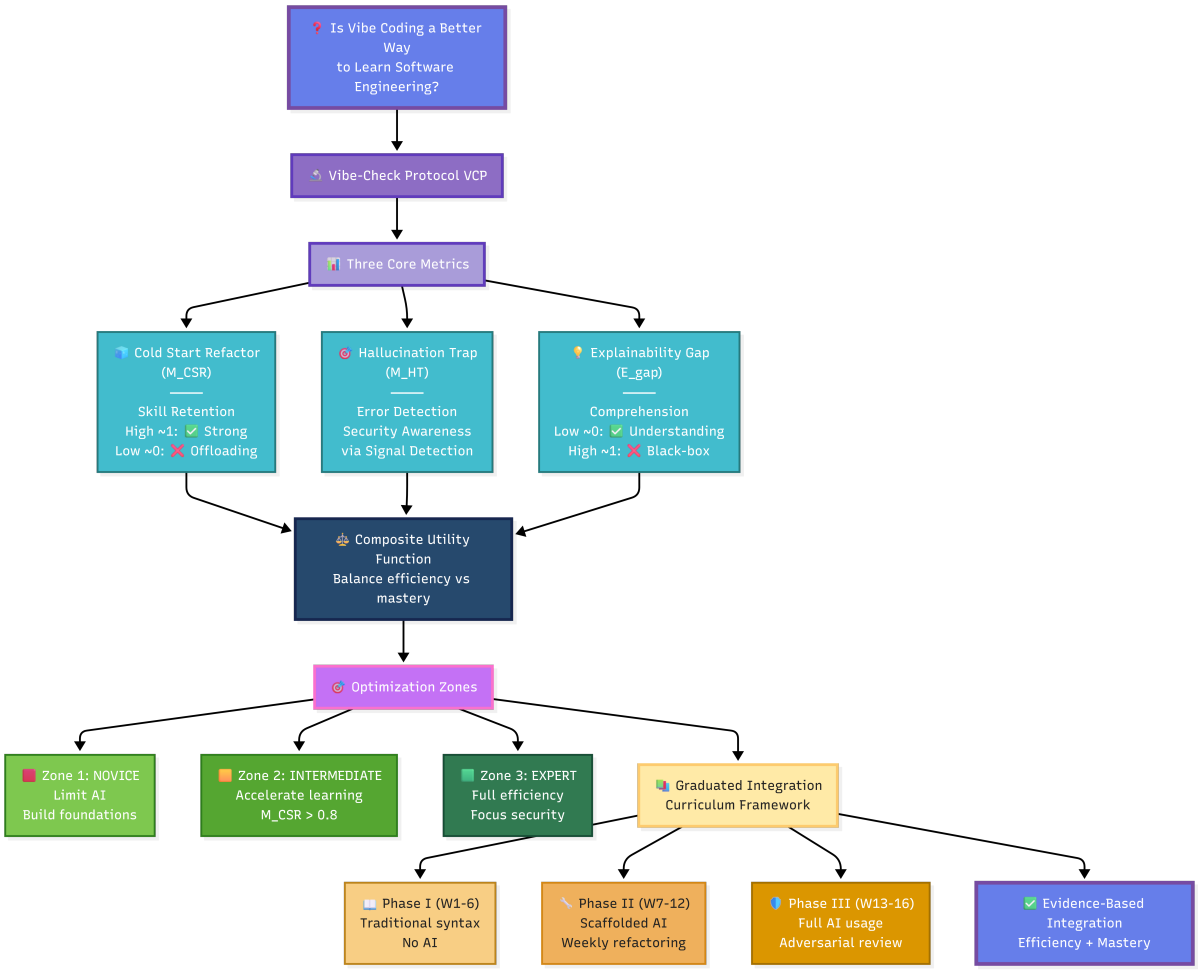

图 1：**Vibe-Check 协议（VCP）框架**。实验设计采用混合方法，将传统教学（对照组）与氛围编程（实验组）进行比较。该框架通过三个有理论依据的指标量化教育影响：(1)冷启动重构(M_{CSR})，通过程序性衰退衡量技能保持；(2) 幻觉陷阱检测(M_{HT})，运用信号检测理论评估警觉性和错误敏感性；以及 (3) 可解释性差距(E_{gap})，衡量生成代码复杂度与学生理解之间的元认知脱节。

3.1 理论基础与实验设计

Vibe-Check 协议以三个互为补充的理论框架为基础，共同涵盖编程教育中的认知、发展和元认知维度。认知负荷理论 [Sweller, 1988] 提供了基础性认识，将认知负荷区分为三类：代表任务本身固有复杂性的内在认知负荷、由教学设计不佳导致的外在认知负荷，以及有助于构建深度学习所必需的认知图式的促进性认知负荷。在这一理论背景下，VCP 指标专门用于评估人工智能辅助是否会无意中降低促进性认知负荷，从而可能损害学生在应对具有挑战性的编程概念时，通过富有成效的努力所经历的深度学习过程。鉴于人工智能辅助编程看似轻松，可能掩盖了技能习得所必需的认知加工缺失，这一问题尤为值得关注。

作为对这一认知视角的补充，专长发展研究 [Ericsson 等人,1993] 强调，任何领域的专家级表现都需要大量刻意练习，其特点是即时反馈、难度逐步提升，以及长期持续投入。VCP 框架通过评估 Vibe Coding 环境是否提供技能习得所必需的“适当困难”，还是通过自动化编程中原本能够促进有意义学习和技能发展的环节，彻底绕过这些挑战，从而将这些原则付诸实践。这项评估至关重要，因为人工智能辅助带来的效率增益，可能是以牺牲程序性知识和问题解决技能为代价，而这些正是编程能力的核心。

第三个理论支柱源自元认知监控研究 [Kruger 和 Dunning, 1999], 该研究探讨感知能力与实际能力之间往往存在问题的校准。相关研究表明，在某个领域能力较低的人往往最容易对自身能力过度自信；这一现象与人工智能辅助学习环境直接相关，因为功能性代码生成可能造成一种理解上的错觉。VCP 框架通过“可解释性差距”指标应对这一挑战，该指标对自我评估的理解程度与实际展示出的代码功能和设计原则理解程度进行定量比较。这种全面的理论整合确保 VCP 能够捕捉学习过程中的多个关键维度，而非依赖可能遗漏教育影响重要方面的单一结果指标。

VCP 的实验实施采用了精心设计的混合方法，将严谨的定量指标与深入的定性访谈相结合，并以 Rojas-Galeano [Rojas-Galeano, 2025]确立并验证的两阶段方案为基础。拟议研究中的参与者将通过分层随机分配程序，根据其既往编程经验进行仔细匹配，并系统地分配至不同实验条件：一组采用氛围编程方法，传统对照组则接受以语法为重点的常规教学。纵向设计贯穿整个学期的多个教学环节，能够全面分析技能发展轨迹，并识别不同教学方法的即时与延迟效应。

样本量确定遵循教育研究中常用的成熟统计效能分析程序，计算时以 80% 的统计效能为目标，以检测中等效应量（Cohen’s $d = 0.5$），显著性水平为 $\alpha = 0.05$。这些参数表明，被试间比较每组至少需要约 64 名参与者，被试内重复测量设计则需要 34 名参与者。然而，考虑到纵向教育研究中约 20% 的预期流失率，为确保整个研究期间具备充足的统计效能，建议每种实验条件招募 100 名参与者。

任务校准采用全面的多维方法，确保难度评估有效，并能进行有意义的跨条件比较。圈复杂度是衡量控制流复杂度的主要量化指标，可不受主观因素影响，客观评估任务难度。专家小组评分则通过结构化评估对此方法进行补充，

评估采用经验证的三点评分量表，难度从基础到高级不等，并将评定者间一致性系数维持在 $\kappa > 0.75$ 以上，以确保信度。此外，先修概念枚举会系统地识别所需的知识要素，并将其对应到具体的课程学习目标，确保任务复杂度与课程要求及学生的准备程度相匹配。所有实验条件下的人工智能工具均保持一致，例如通过 Cursor 界面使用 Claude 4 Sonnet，并全面记录所有人机交互，以便事后详细分析使用模式、提示词有效性和迭代优化策略。

3.2 指标 1：冷启动重构 (M_{CSR})

冷启动重构通过对移除人工智能脚手架后程序性知识的衰减进行建模，评估技能保持情况。我们认为，技能保持遵循指数衰减函数，与艾宾浩斯遗忘曲线 [Ebbinghaus, 1913], 类似，也符合认知负荷理论对图式习得的强调 [Sweller, 1988]。

3.2.1 数学表述

令 $S(t)$ 表示初次借助人工智能完成实现后，在时间 t 时保留下来的技能水平。我们将其建模为：

$$S(t) = S_0 \cdot e^{-\lambda t} \tag{1}$$

其中， S_0 是初始功能熟练度（对于可运行的代码，通常为 1.0）， λ 是衰减常数。

在 VCP协议中，我们会测量在时间 $t = \Delta t$ （例如 24 小时）进行无辅助重构任务时的重构速度(V_{rec})。指标 M_{CSR} 定义为无辅助重构速度与初始人工智能辅助构建速度之比，并按任务复杂度进行缩放，以补偿人工智能生成在复杂任务中固有的速度优势：

$$M_{CSR} = \frac{V_{rec}}{V_{build}} \cdot \Omega(C) \tag{2}$$

其中，$\Omega(C)$ 是一个复杂度加权函数，根据代码的圈复杂度（CC）和 Halstead体积（V）推导得出：

$$\Omega(C) = \alpha \cdot \ln(CC) + \beta \cdot V \tag{3}$$

此处， α 和 β 是根据专家级人工开发者的基线数据得出的系数。 $M_{CSR} \approx 1$ 表示学生已充分内化相关逻辑，能够以专家级水平将其复现，说明其保持程度较高。接近 0 的数值则表示存在严重的认知卸载。

3.3 指标 2：幻觉陷阱检测（M_{HT}）

幻觉陷阱检测评估学生的警觉性和代码审查能力。我们使用信号检测理论（SDT） [Green 等人， 1966] 对其进行建模，以区分对错误的真实敏感度与单纯的反应偏差（例如，盲目接受或盲目拒绝代码的学生）。

3.3.1 数学表述

我们将学生的敏感度指数 d' 定义为信号（错误）与噪声（正确代码）分布之间的距离：

$$d' = Z(\text{Hit Rate}) - Z(\text{False Alarm Rate}) \tag{4}$$

其中 $Z(\cdot)$ 是标准正态分布的逆累积分布函数 n.

- **命中率（H）**：正确识别注入漏洞（例如 SQL 注入、竞态条件）的概率。

- **虚警率（F）**：将正确代码误判为有错误的概率。

指标 M_{HT} 通过以职业胜任力阈值 δ为中心的 S 形函数归一化到 $[0, 1]$ 尺度：

$$M_{HT} = \frac{1}{1 + e^{-k(d'-\delta)}} \tag{5}$$

其中， k 决定了过渡的陡峭程度。这一表述会惩罚仅靠猜测取得高检出率的学生（这会增加虚警），并奖励能够准确识别细微逻辑缺陷的学生。

3.4 指标 3：可解释性差距（E_{gap}）

可解释性差距量化了生成代码的复杂程度与学生概念理解之间的脱节。我们采用信息论方法[Shannon， 1948], 比较代码的熵与学生解释的熵。

3.4.1 数学表述

令 $H(C)$ 表示代码的控制流图的香农熵，即完整描述其行为所需的信息量。令 $H(E)$ 表示学生解释的语义熵，通过与真实本体进行概念映射计算得出。

可解释性差距定义如下：

$$E_{gap} = 1 - \frac{H(E)}{H(C) + \epsilon} \tag{6}$$

其中 ϵ 是一个较小的正则化项。

- 如果 $H(E) \approx H(C)$，那么 $E_{gap} \to 0$，表明学生的心智模型与代码复杂度相匹配。

- 如果 $H(E) \ll H(C)$，那么 $E_{gap} \to 1$，这表明存在“黑箱”使用情况，即代码包含学生无法阐述的逻辑。

3.5 综合评分与优化

为回答研究问题“氛围编程更好吗？”，我们定义一个教学效用函数 U：

$$U = w_1 \cdot M_{CSR} + w_2 \cdot M_{HT} + w_3 \cdot (1 - E_{gap}) - \gamma \cdot T_{dev} \tag{7}$$

其中，T_{dev} 表示开发时间（氛围编程的主要优势）。该方程使我们能够计算<style id='2'>盈亏平衡点：即效率增益（$\gamma \cdot T_{dev}$）恰好足以抵消保持率或安全意识潜在损失的阈值。

数据分析采用混合效应模型：

$$Y_{ij} = \beta_0 + \beta_1 \cdot \text{Condition}_{ij} + \beta_2 \cdot \text{Time}_{ij} + u_j + \epsilon_{ij} \tag{8}$$

从而能够在控制每位学生的个体差异（u_j）的同时，分离出 Vibe Coding 干预（β_1）的具体影响。

4 理论启示与预期效用

4.1 优化边界：何时采用氛围编程?

本研究的首要目标不仅是批判氛围编程，还要构建一个用于优化部署氛围编程的决策理论框架。我们的数学建模表明，氛围编程的效用与学习者专业水平之间呈非线性关系。

我们提出**认知负荷优化边界**这一理论构想，该边界由技能衰退曲线（$S(t)$）与复杂度管理曲线的交点定义，用以划定针对不同学生专业水平的最优教学策略。边界这一概念认识到，氛围编程的教育价值会因学习者当前的能力水平而有显著差异，并指出在所有技能水平上统一应用可能并非最优。

在**基础习得区**，其中初学程序员的初始技能水平 S_0 仍相对较低，引入氛围编程方法通常会导致技能迅速衰退，趋近于 $\lambda \to \infty$，实际上意味着程序性知识的立即丧失。在这一发展阶段，起初看似与语法掌握和底层实现细节相关的外在认知负荷，实际上是促进性认知负荷，对于基础图式形成和概念理解至关重要。与语法、调试和基础算法思维的斗争，是一种生产性困难，有助于构建后续编程能力所必需的基础心智模型。因此，在此阶段应严格限制 Vibe Coding 干预，

以保留通过直接操作代码而产生的关键学习过程。

当学生达到中级能力水平时，**架构探索区** 便会出现。具体而言，当冷启动重构指标（M_{CSR}）稳定在 0.8 以上时，表明学生已充分保持了基础编程概念。在这一发展阶段，氛围编程会转变为强大的教学加速器，使学生能够运用高阶抽象，并应对日益增加的代码复杂度（$H(C)$），而不会被实现细枝末节压得不堪重负。在该区域内，可解释性差距（E_{gap}）是主要的控制变量和监测机制。只要 E_{gap} 保持在 0.3 以下，即学生理解与代码复杂度基本相符，氛围编程方法就能提升学习成果，大幅拓展可完成编程项目的范围与复杂程度，从而让学生接触到架构模式和系统级思维；若受时间和复杂度限制，这些内容原本难以触及。

最后，**专业效率区** 面向能力接近专家水平的高级学生。对这类学生而言，教育重点发生根本转变：从基础编程能力转向批判性评估和质量保证技能。对于这些学习者，幻觉陷阱检测（M_{HT}）成为最关键的评估指标，因为主要风险已从“不会编程”转变为“无法识别人工智能生成的解决方案中细微却重大的缺陷”。该区域强调培养专业级代码审查能力、安全意识，以及有效与 AI 系统协作所需的成熟判断力，同时保持工程标准，并检测功能测试可能遗漏的潜在漏洞或逻辑错误。

4.2 教学实践与课程设计的启示

Vibe-Check 协议在理论上的应用及其预期的实证验证，对当代计算机科学教育中的教学实践具有若干重要启示，从根本上挑战了人工智能辅助学习工具能够始终带来积极教育成果这一假设。该框架的核心发现表明，氛围编程并非天然有益或有害的教育方法，而是一种教学工具，其价值在很大程度上取决于前文所述的认知优化边界。这种细致入微的认识要求教育者不再对人工智能工具采用做简单的二元判断，而是转向更成熟、注重情境的整合策略，兼顾学生发展阶段、学习目标和长期能力目标。

VCP 指标所提供的系统化评估能力，使基于证据的课程设计成为可能，从而在人工智能辅助开发毋庸置疑的效率优势与有据可查的技能衰退和浅层理解风险之间取得平衡。实证证据持续表明，氛围编程能显著加快初始代码产出，并使学生在学习早期就能参与更复杂的项目；与此同时，该框架也揭示，传统教学方法在基础技能保持方面表现更优，这一点由冷启动

重构指标（M_{CSR}）以及通过幻觉陷阱检测（M_{HT}）评估的关键错误检测能力，尤其是在奠定基础能力的早期发展阶段。

近期基准测试研究（如 SUSVIBES [Zhao 等人， 2025] ）记录的质量与安全维度，使这些教学考量更显紧迫：相当比例功能正确的人工智能生成的解决方案，都含有细微但可能造成严重后果的潜在缺陷和安全漏洞。仅通过氛围编程方法接受训练、却未同步培养批判性评估技能的学生，毕业时或许拥有令人印象深刻的作品集项目，却缺乏在专业软件开发情境中必需的成熟审查能力。这一发现与新兴的行业共识高度一致：该共识区分了不加批判地接受人工智能输出的随意“氛围编程”，以及始终坚持严格人工监督和质量保证标准的专业“人工智能辅助工程” [Osmani， 2025]。

4.3 建议的课程调整

基于 VCP 指标提供的理论框架和实证见解，我们提出一种**分阶段整合框架** ，通过三个精心设计的发展阶段，系统地引入人工智能辅助编程方法；每个阶段都针对具体的学习目标和能力基准。该框架认为，要取得最佳教育成果，需要策略性地安排人工智能整合的时机并提供支架式教学，而非在所有学习情境中一概采用。

最初的**语法与语义阶段**为期六周，采用传统教学方法，并严格限制人工智能辅助，以确保学生掌握基础技能。在这一奠定基础的关键时期，主要教学目标是尽量缩小较简单代码单元的可解释性差距（E_{gap}），确保学生全面理解基本的编程结构、控制结构和算法思维模式。在此阶段暂时禁止使用人工智能工具，旨在消除可能妨碍必要图式形成的认知卸载风险，要求学生直接学习语法、调试过程和基本问题解决策略，这些都是编程能力的基石。

接下来的**支架式加速阶段**从第七周持续到第十二周，在严格管控的情境下引入氛围编程方法，尤其针对样板代码生成和自动化测试框架，同时维持人工监督并满足理解要求。在这一中间阶段，核心目标是让冷启动重构表现维持在临界阈值$M_{CSR} > 0.8$以上，以表明学生具备足够的技能保持能力，能够支撑更高级的学习目标。为确保学生继续投入核心编程概念的学习，他们必须每周完成手动重构练习，系统地修改、扩展并改进人工智能生成的代码片段，从而在受益于加速开发能力的同时，展现对相关内容的理解。

The c 最终的**批判性审查阶段**为第十三至第十六周， r-

允许全面采用氛围编程方法，同时将教学重点转向最大限度地提升幻觉陷阱检测能力（M_{HT}）并培养专业级的质量保证技能。在这一高级阶段，学生担任技术审计员，负责系统评估 AI 智能体的输出，重点识别逻辑错误、安全漏洞，以及 SUSVIBES 等综合基准测试中记录的细微缺陷 [Zhao 等人， 2025]。这种方法使学生从人工智能生成代码的被动使用者，转变为能够运用 AI 能力、同时保持批判性监督并遵循专业工程标准的积极协作者。

4.4 专业区分：人工智能辅助工程与氛围编程

如第 2.1 节所述，业界讨论区分了随意的“氛围编程”和严谨的“人工智能辅助工程”。氛围检查协议将这一差异转化为可操作的评估标准，使我们能够系统地评估学生是否正在培养后者所需的批判性思维和质量保证能力，而非仅仅成为高效消费 AI 生成内容的用户。

在幻觉陷阱检测指标（M_{HT}）上表现优异，可作为量化指标，表明学生正在以真正的人工智能辅助工程师的方式工作，而非被动接受自动化解决方案。这些学生展现出所需的警觉性、技术判断力和系统性评估技能，能够识别细微的逻辑错误、潜在的安全漏洞以及设计缺陷，而这些问题仅靠功能测试可能无法发现。相反，较低的可解释性差距值（E_{gap}）则表明学生真正掌握并理解了相关概念，远不止于表面上接受可运行的代码。这些指标共同区分了两类学生：一类能够在遵循工程标准的同时与 AI 系统有效协作；另一类则形成了依赖关系，而这种关系在职业环境中可能带来问题。

这一差异回应了近期实证研究记录的一项关键关切：学生的偏好和自我评估表明，即使身处 AI 增强型开发环境，他们也意识到保持基础编程技能的重要性。Rojas-Galeano 报告称，约 60% 的受访学生明确同意“需要编程技能来调整 AI 输出” [Rojas-Galeano，2025]，这表明学生自身也认识到完全依赖 AI 的方法存在局限。VCP 框架提供了必要的系统化评估机制，确保课程能够支持学生的这些志向，将研究记录的“能力错觉”转化为可衡量、可验证的掌握程度，并使之符合教育目标和职业要求。

4.5 定向干预策略与补救方法

Vibe-Check 协议的诊断能力，使教育工作者能够依据学生在三项核心指标上的具体表现概况，开发并实施有针对性的教育干预措施，超越一刀切的方法，转而采用个性化补救策略，以满足每位学生的学习需求并弥补其技能差距。根据冷启动重构指标测得的衰减率较高的学生，将受益于系统性的间隔

检索练习干预，其中纳入间隔逐步延长的无人工智能辅助编码练习，从而强化程序性记忆巩固，并减少对外部支架的依赖。这些干预措施可以包括每周进行的无辅助重构挑战、复杂度逐步提高的限时编码练习，以及结构化反思活动，帮助学生将最初由人工智能辅助支持的问题解决策略内化。

对于在幻觉陷阱检测中敏感度得分较低的学生，明确的代码审查培训至关重要，重点应放在系统接触有据可查的反模式、常见漏洞类型，以及常被自动化检测系统忽略的隐蔽缺陷类别上。此类培训应着重讲解真实世界中的失效模式，包括利用竞态条件的时序攻击、影响数据库和系统安全的各种注入漏洞，以及危及访问控制的身份验证绕过技术。课程应纳入真实安全事件的案例研究、对易受攻击代码示例的结构化分析，并让学生亲自使用专业开发环境中常见的静态和动态分析工具。

除了补救之外，VCP 框架还支持识别和培养积极的学习轨迹，利用学生已有的优势，同时针对需要改进的方面进行提升。初步观察表明，随着学生逐渐形成更精准、语境更丰富的沟通策略，与 AI 系统协作时的提示工程质量会显著提高，这表明他们对如何有效地与自动化工具协作有了更深入的理解，元认知也随之发展。同样，随着氛围编程方法使学生能够更早接触系统级设计模式和复杂的软件架构（这些内容原本会因实现复杂度的障碍而难以涉足），其架构设计水平似乎也有所提升。这些积极进展支持对人工智能辅助学习采取平衡的看法：氛围编程虽会带来有据可查、需要谨慎管理的风险，但在得到适当引导并接受系统性监测时，也能创造独特的教育机会，提升而非损害学习成果。

5 局限性与未来研究

5.1 当前局限性与方法论限制

Vibe-Check 协议虽然代表着评估人工智能辅助编程教育的现有方法取得了重大进展，但仍面临若干重要局限，未来研究需要认真承认并考虑这些问题。当前这项工作的根本限制在于其理论地位：VCP 框架仍有待通过受控课堂实验获得全面的实证验证。这一局限源于机构审查委员会的要求、纵向教育研究所需的较长周期，以及人工智能工具迅速演进所形成的动态研究对象等因素的复杂交织。由于缺乏实证验证，提出的阈值、权重和数学表述仍需通过真实环境中的实施加以确认，才能充分确立其实际效用。

Gener 可推广性方面的限制是另一个重要的方法论问题，因为未来 e

试点研究必然会涉及来自特定院校和文化背景的学生，他们可能无法代表编程教育环境的广泛多样性。拟议阈值和指标权重对不同群体的适用性仍有待通过实证研究确立，这些群体包括追求职业转型的非传统学生、教育背景各异的国际学生群体，以及具有不同动机结构的职业再培训情境中的群体。这些群体差异可能会显著影响 VCP 指标的效度，以及在不同教育环境中对结果的解读。

效度面临的方法论威胁还包括潜在的霍桑效应，即观察和测量过程可能会无意中改变参与者行为，从而损害评估结果的真实性。对于幻觉陷阱检测指标（M_{HT}）而言，这一问题尤为突出：学生意识到自己正在接受错误检测评估，可能会因警觉性提高而人为抬高检测率，但这种警觉性在自然的学习环境中未必能够持续。因此，未来的实施必须探索更完善的非侵入式测量策略和对照条件，在维护评估过程完整性的同时，减轻这些观察偏差。

人工智能能力的快速演进，为 VCP 指标的验证与应用增添了另一层复杂性。人工智能工具及其能力正迅速发展，这意味着当前接受评估的具体系统和性能特征，可能与之后的学期或年份所能使用的系统和性能特征大相径庭。技术的这种动态变化要求随着模型能力的进步，持续校准和验证 VCP 指标，并不断适配框架，以确保测量始终切合实际、准确无误，并与当前人工智能工具的能力和局限保持一致。

资源密集度是完整 VCP 框架在教育环境中广泛应用的一项重大实际障碍。全面实施需要大量投入，用于聘请经过培训、能够持续可靠地开展评估的人工评审员，建设复杂的纵向跟踪基础设施以管理跨学期数据收集，并投入相当多的教师时间进行实施和持续监测。在考虑将其应用于大规模招生课程或资源受限的教育机构时，规模化挑战尤为突出。如果不开发更多自动化评估工具，或大幅简化相关流程，VCP 的实用价值可能仍局限于研究领域，难以产生更广泛的教学影响。此外，当前框架中，安全意识与教学评估的整合仍不完善。虽然 VCP 通过“幻觉陷阱检测”指标纳入了安全考量，但该框架尚未全面衔接专业安全评估标准和行业漏洞检测培训流程。这一差距意味着错失了更好地将教育评估与职业准备相结合的机会；尤其考虑到安全意识在软件开发职业中的重要性日益提高，且近期基准测试研究已指出人工智能生成的代码中存在漏洞。

5.2 未来研究方向与方法论扩展

Vibe-Check 协议的开发与验证为未来研究开辟了诸多方向，既能应对当前局限，也能拓展框架的范围与适用性。跨机构验证是最紧迫、最关键的研究优先事项，需要协调开展多地点研究，涵盖来自不同机构背景、地理区域和文化背景的多样化学生群体。此类全面验证工作对于建立常模表现数据，并确定拟议阈值和权重在不同教育环境中的普适性至关重要。这些研究应系统考察班级规模、教师经验、技术基础设施和教学传统等机构因素如何影响 VCP 指标的表现与解读。

整合先进的生理和认知评估指标，为深入洞察 AI 辅助编程教育背后的学习过程提供了可喜的机会。系统地纳入眼动追踪技术、结构化出声思维协议和神经影像学方法，可增强对认知参与模式的观察能力，而这些模式远超可观察到的行为结果。此类多模态评估策略有助于研究人员区分真正的概念理解与表层的模式匹配行为，并有望揭示当前行为测量指标无法检测到的认知过程。更深入的理解有助于更细致地解读 VCP 指标，并识别可补充现有测量指标的其他评估维度。

要应对当前局限中发现的可扩展性挑战，需要大力投入自动化评估工具开发，利用机器学习和自然语言处理方面的进展，降低实施 VCP 所需的人力资源。通过对学生解释进行复杂的语义分析，自动化方法有望完成可解释性差距 (E_{gap}) 编码，生成针对特定代码上下文量身定制的恰当幻觉陷阱检测场景，并提供实时反馈机制，以支持学生学习和教师决策。此类工具的开发将使 VCP 从一种耗费大量资源的研究工具转变为实用的课堂评估系统，从而适合广泛应用。

追踪职业成果的纵向研究是确立 VCP 评估的预测效度和现实相关性的另一项重要研究方向。对毕业生进入行业岗位后的情况进行多年追踪，将为了解 VCP 表现与后续职业胜任力、职业发展及长期技术贡献之间的关系提供关键证据。此类研究可以验证 VCP 指标上的高表现是否能在专业软件开发情境中转化为可衡量的优势，从而确立该框架在教育评估和职业准备评估中的效用。

通过系统开展随机对照试验来优化干预措施，有助于确定针对不同学习者特征和教育情境最有效的教学策略。比较不同支架式教学方法、时机策略和教学调整的研究，可以确定最佳实施参数

以适用于不同学生群体的分阶段整合框架。这些研究应系统地改变无人工智能教学的持续时间、过渡阶段中的支架式活动类型，以及评估任务的复杂度递进等因素，从而确定将人工智能整合到编程教育中的循证最佳实践。

文化适应研究是 VCP 验证中一个至关重要但目前尚未得到充分探索的维度，因为不同国际环境中的技术态度、协作规范、教育传统和学习偏好存在显著差异。了解文化因素如何调节 VCP 的研究发现并影响对各项指标的解读，或可揭示该框架适用的重要边界条件，并为其在全球范围内的实施提出必要的调整方向。此类研究可以考察个人主义与集体主义文化取向如何影响协作式人工智能使用，不同教育传统如何塑造学生对技术辅助的预期，以及对权威和专业知识的不同态度如何影响学生对人工智能生成的解决方案进行批判性评估。此外，加强 VCP 框架中的安全培训整合，有助于强化教育评估与网络安全意识开发环境中的职业准备之间的联系。未来研究应开发全面的课程模块，系统地将 VCP安全意识测量指标与成熟的专业安全编码实践相结合，并可纳入类似于 SUSVIBES 基准测试所采用的仓库级挑战 [Zhao 等人， 2025]。对安全问题的这一拓展关注，或可使 VCP 从通用的编程教育评估工具转变为一个专门框架，帮助学生为在安全攸关的开发环境中有效、安全地使用人工智能辅助做好准备。

6 结论

Vibe-Check 协议为评估人工智能辅助编程的教育影响提供了一个系统性框架。通过量化技能保持、错误检测和概念理解，VCP 回应了氛围编程时代对基于证据的评估的迫切需求。该框架超越了简单的功能正确性评估，能够捕捉学习的认知维度，区分真正掌握与能力错觉。

归根结底，VCP 的目标是引导人工智能融入教育。通过明确人工智能加速所能支持而非取代深度学习的教学边界，教育工作者可以培养新一代软件工程师，让他们善用人工智能这一强大工具，同时保有构建稳健、安全且富有创新性的系统设计所必需的基础专业能力。

参考文献

赫尔曼·艾宾浩斯。《记忆：对实验心理学的贡献》。哥伦比亚大学师范学院，1913年。网址 https://doi.org/10.1037/10011-000。Mihail Eric。CS146S：现代软件开发者。斯坦福大学课程目录，2025年。网址 https://themodernsoftware.dev/。K Anders Ericsson、Ralf T Krampe 和 Clemens Tesch-R¨omer。《刻意练习在专家表现习得中的作用》。《心理学评论》，100(3):363，1993年。网址 https: //psycnet.apa.org/doi/10.1037/0033-295X.100.3.363。David Marvin Green、John A Swets 等。《信号检测理论与心理物理学》，第1卷。威利，纽约，1966年。网址 http://andrei.gorea.free.fr/Teaching_fichiers/SDT% 20and% 20Psytchophysics.pdf。Andrej Karpathy。氛围编程。Twitter/X，2025年2月。网址 https://x.com/karpathy/ status/1886192184808149383。最初提出“氛围编程”一词的推文。

Justin Kruger 和 David Dunning。无能且不自知：难以识别自身能力不足如何导致自我评估过高。《人格与社会心理学杂志》，77(6):1121，1999。网址：https://psycnet.apa.org/doi/10.1037/0022-3514. 77.6.1121。

Sam Lau 和 Philip Guo。从“在理解之前先禁止”到“抵制徒劳”：随着越来越多学生使用 ChatGPT 和 GitHub Copilot 等人工智能代码生成与解释工具，大学编程教师计划如何适应。载于*2023*年*ACM*国际计算教育研究会议论文集（第*1*卷），第106–121页，2023。网址：https://dl.acm.org/doi/10.1145/3568813.3600138。

Ference Marton 和 Roger S¨alj¨o。论学习中的质性差异：I——结果与过程。《英国教育心理学杂志》，46(1):4–11，1976。网址：https://doi.org/10.1111/j.2044-8279.1976.tb02980.x。

Addy Osmani。《氛围编程》不同于人工智能辅助工程。在线文章，2025 年。URL https://medium.com/@addyosmani/vibe-coding-is-not-the-same-as-ai-assisted-engineering-3f81088d5b98。Hammond Pearce、Baleegh Ahmad、Benjamin Tan、Brendan Dolan-Gavitt 和 Ramesh Karri。键盘前睡着了吗？评估 GitHub Copilot 的代码贡献安全性。《*ACM* 通讯》，68(2)：96–105，2025 年。URL https://dl.acm.org/doi/10.1145/ 3610721。Evan F Risko 和 Sam J Gilbert。认知卸载。《认知科学趋势》，20(9)：676–688，2016 年。URL https://doi.org/10.1016/j.tics.2016.07.002。Sergio Rojas-Galeano。课堂上的新生：探索学生对人工智能编程助手的认知。*arXiv* 预印本 *arXiv:2507.22900*，2025 年。URL https://arxiv.org/abs/2507. 22900。

Claude E Shannon。通信的数学理论。《贝尔系统技术期刊》，27(3)：379–423，1948 年。URL https://doi.org/10.1002/j.1538-7305.1948.tb01338.x。

John Sweller。《认知负荷与问题解决：对学习的影响》。《认知科学》，12（2）：257–285，1988。网址：https://doi.org/10.1207/s15516709cog1202_4。

Xingyao Wang、Boxuan Li、Yufan Song、Frank F Xu、Xiangru Tang、Mingchen Zhuge、Jiayi Pan、Yueqi Song、Bowen Li、Jaskirat Singh 等。《OpenHands：面向通用智能体的人工智能软件开发者开放平台》。*arXiv* 预印本 *arXiv:2407.16741*，2024。网址：https: //arxiv.org/abs/2407.16741。

John Yang、Carlos E Jimenez、Alexander Wettig、Kilian Lieret、Shunyu Yao、Karthik Narasimhan 和 Ofir Press。《SWE-Agent：智能体—计算机接口助力自动化软件工程》。《神经信息处理系统进展》，37：50528–50652，2024。网址：https://dl.acm.org/doi/10.5555/3737916.3739517。

Songwen Zhao、Danqing Wang、Kexun Zhang、Jiaxuan Luo、Zhuo Li 和 Lei Li。《氛围编程安全吗？对现实世界任务中智能体生成的代码的漏洞进行基准测试》。*arXiv* 预印本 *arXiv:2512.03262*，2025。网址：https://arxiv.org/abs/2512.03262。