# 为机器编写代码，而不仅仅是为人类：使用代码健康度指标量化人工智能友好性

Markus Borg CodeScene
和隆德大学 瑞典马尔默
m
arkus.borg@codescene.com

Nadim Hagatulah 隆德
大学 瑞典隆德
nadi
m.hagatulah@cs.lth.se

Adam Tornhill CodeScene
瑞典马尔默
ada
m.tornhill@codescene.com

Emma Söderberg 隆德
大学 瑞典隆德
e
mma.soderberg@cs.lth.se

arXiv:2601.02200v1 [cs.SE] 5 Jan 2026

## 摘要

我们正步入一个混合时代，人类开发者与人工智能编程代理在同一个代码库中协同工作。长期以来，工业界一直致力于优化代码，使其便于人类理解；如今，确保能力各异的大语言模型能够可靠地编辑代码，正变得越来越重要。本研究通过对竞赛编程领域的 5,000 个 Python 文件组成的数据集进行基于大语言模型的重构，探讨“人工智能友好代码”这一概念。我们发现，针对人类理解校准的质量指标代码健康度，与人工智能重构后的语义保持之间存在显著关联。研究结果证实，人类友好型代码也更适合 AI 工具。这些结果表明，各组织可以利用代码健康度来判断哪些地方采用人工智能干预的风险较低，以及哪些地方需要加强人工监督。投资于可维护性不仅有助于人类，也能为大规模应用人工智能做好准备。

## CCS 概念

• **软件及其工程 → 软件维护工具**。

## 关键词

软件可维护性、代码质量、重构、人工智能助手

**ACM 引用格式：**
Markus Borg、Nadim Hagatulah、Adam Tornhill 和 Emma Söderberg。2026。为机器编写代码，而不仅仅是为人类：使用代码健康度指标量化人工智能友好性。发表于。 ACM，美国纽约州纽约市，12 页。
https: //doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 引言

几十年来，人们始终坚信“程序必须编写得便于人们阅读，而机器执行只是顺带的目的” [2]。易于人类阅读的代码对于维护安全、可靠且高效的软件开发至关重要 [11, 30, 37]。但随着人工智能辅助编程的兴起，源代码如今面向更广泛的受众：机器也需要理解其意图。

Thoughtworks[1] 收集的早期实地观察表明，人工智能辅助编程工具在代码质量较高时表现更好。具体而言，结构良好、模块化的代码似乎能降低产生幻觉的风险，并带来更准确的建议。他们在2025年4月的《技术雷达》中将此称为“*AI* 友好型代码设计”，并探讨了迄今为止公认的最佳实践如何与人工智能友好性相契合 [43]。如果这一观察成立，就意味着，为便于人类理解而优化的代码，也更容易被大语言模型（LLM）处理和演进。

人工智能友好性的影响深远：截至2025年，约80%的开发者已在工作中使用人工智能工具 [31],，预计其采用率还将迅速增长。Gartner预测，这一比例将从2024年初的14%升至 2028 [46]时的90%。与此同时，据报告，不到10%的组织会系统地跟踪技术债务 [27], ；研究还发现，糟糕的代码质量会使开发者浪费多达42%的时间 [42]。这些数据表明，如今大量生产代码的结构可能并不适合人工智能可靠地介入，从而增加缺陷和高昂返工的风险。

在本文中，我们研究代码质量与人工智能友好性之间的关系。我们使用代码健康度（CH）指标衡量代码质量；此前的研究已验证，该指标能够预测缺陷和开发工作量 [6, 9, 44]。近期面向工业界的人工智能研究也采用了这一指标 [8, 29]。我们以人工智能生成的重构成功率作为人工智能友好性的代理指标。重构是指在不改变代码行为的前提下改进现有代码的设计。重构使我们能够将通过的单元测试作为功能正确性的判定依据。因此，如果测试通过，则人工智能重构是正确的；如果 CH 提高，则是有益的 。

我们的结果证实，人类友好型代码与 AI 工具的兼容性更高。对于处于健康代码健康度范围内的代码（$CH \geq 9$），接受重构任务的大语言模型的破坏率显著更低，相应的相对风险降低了 15%–30%。此外，我们还表明，在预测重构正确性方面，CH 优于困惑度 （PPL，一种大语言模型内在置信度指标）和源代码行数（SLOC）。

这些发现有助于软件组织采用人工智能辅助编程。我们建议使用代码健康度（CH）来识别适合进行 AI 处理的代码部分，同时指出

允许免费为个人或课堂使用而制作本作品全部或部分内容的数字副本或纸质副本，前提是这些副本不得用于营利或商业利益，且每份副本均须附有本声明和首页上的完整引用。必须尊重本作品中由作者以外的其他方拥有版权的组成部分。注明出处后允许摘录。除此之外，如需复制、再版、发布到服务器或分发至列表，必须事先获得明确许可和/或支付费用。请通过 permissions@acm.org 申请许可。
*Conference'17*，美国华盛顿特区© 2026 版权所有者/作者保留版权。出版权已授权给 ACM。ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://www.thoughtworks.com/

存在过高破坏风险的代码。更广泛地说，这项研究为人工智能应用补上了缺失的一环：一种让人类与机器达成一致的共享代码质量指标。早期研究将代码质量视为一项业务要务，而我们认为，代码质量是安全、有效地使用人工智能的先决条件；未来十年，这或许会成为软件组织的生死攸关之事。

本文其余部分的组织结构如下。第2节介绍背景和相关工作。在第3节中，我们介绍研究问题、数据集和方法。第4节报告研究结果。最后，第5节讨论其影响，第6节总结未来工作的方向。

## 2 背景和相关工作

本节首先介绍代码健康度和困惑度，然后介绍代码理解和人工智能重构方面的相关工作。

### 2.1 可维护性与代码健康度

CodeHealthTM（代码健康度，CH）是 CodeScene 软件工程智能平台中使用的一项质量指标，旨在衡量人类开发者理解代码所需的认知难度。CodeScene 会识别代码异味 [25], , 例如上帝类、深层嵌套逻辑和重复代码。对于本文研究的 Python，CodeScene 可检测 25 种代码异味。

CodeScene 将检测到的代码异味数量及其严重程度综合起来，为每个文件计算 1 到 10 的分数。分数越低，表示人类的认知负荷越高，也就是维护工作量越大。CodeScene 将文件划分为三个代码健康度区间：健康 （$\mathrm{CH} \geq 9$）、警告（$4 \leq \mathrm{CH} < 9$）和警报（$\mathrm{CH} < 4$）。本研究将警告和警报级别的文件统称为不健康。

我们此前已通过一系列研究验证了代码健康度指标。在一项针对人工标注的可维护性数据集 [38],的研究中，我们报告称，与其他指标相比，代码健康度与人类对可维护性的判断更加吻合，且人类专家的平均 [6]。此外，我们还从商业角度验证了该指标，考察了代码健康度与文件级缺陷密度及开发时间之间的关联 [9, 44]。在先前研究的基础上，我们现在探究代码健康度与人工智能友好性之间的关系。

### 2.2 大语言模型置信度与困惑度

已有研究提出了多种指标来评估大语言模型输出的置信度。熵、互信息和困惑度等内部指标可能可用作输出正确性和幻觉风险的代理指标。在软件领域，Sharma 等人 发现，熵和互信息越高，生成代码的功能正确性越低 [39]。Ray 等人 早期的研究也表明，含有缺陷的代码比正确代码具有更高的熵 [34], ；而这项研究又建立在 Hindle 等人 关于代码自然性的奠基性工作之上 [19]。

PPL 最初是为语音识别任务提出的 [21], ，指预测模型在一个序列上的平均熵的指数。熵可表示“平均意外程度”，而 PPL 则将其转化为“有多少种选择”的尺度。它用于衡量大语言模型根据先前词元预测下一个词元时的置信度——分数越高，置信度越低。从数学上讲，设 $t_{1:N}$ 为一个词元序列。平均交叉熵（$H$）为：

$$H = -\frac{1}{N}\sum_{i=1}^{N} \log p(t_i \mid t_{<i}).$$

困惑度为 $\exp(H)$。在本研究中，我们直接从所研究的五个 Hugging Face 模型获取困惑度 [13]。

### 2.3 代码理解与困惑度

Gopstein 等人 创造了“混淆原子”这一术语，用来指代代码中会使人困惑的、细小而独立的模式 [14]。混淆原子是能够引发困惑的最小代码单元。通过受控实验，他们找出了一组会显著提高误解发生率的混淆原子。例如，“赋值作为值”，如 `V1 = V2 = 3;` 以及“逻辑作为控制流”，如 `V1 && F2();`。

在原子概念的基础上，Abdelsalam 等人讨论了当前混合开发环境中人类程序员与大语言模型在理解方面存在差异所带来的风险 [1]。他们认为，如果人类和大语言模型会被代码中的不同特征所困扰，就存在不一致的风险。通过比较大语言模型的困惑度和脑电图反应，他们发现大语言模型和人类在处理代码时会遇到类似的问题。Casulnuovo 等人此前进行的一项受控实验也表明，人类理解与代码片段对大语言模型而言有多出乎意料之间存在关联 [12]。

Kotti等人探讨了不同编程语言的困惑度 [23]。他们根据 GitHub 项目中代表不同语言的 1,008 个文件，发现强类型语言与动态类型语言之间存在系统性差异。例如，在不同大语言模型中，Perl 的困惑度较高，而 Java 的困惑度较低。他们推测，困惑度的研究结果可用于评估基于大语言模型的代码补全在特定项目中的适用性。本研究探讨了健康与不健康的 Python 代码的困惑度概念。

### 2.4 人工智能重构

与许多软件工程应用一样，自动化代码重构研究也因大语言模型的出现而发生了转变。多项研究探讨了大语言模型与提示词的不同组合能否有效改善设计并消除代码异味。Pomian等人 推出了工具 EM-Assist [33]，用于自动建议并执行提取方法重构。同一团队还持续研究更复杂的重构操作移动方法 [4]。Tornhill等人开发了工具 ACE，用于自动消除 CodeScene 中的五种代码异味 [45]。

2025年，智能体人工智能已成为工业界和研究领域的一大趋势。编码智能体由大语言模型驱动，但通常声称能够：1）理解超出有限上下文窗口范围的代码库；2）保持记忆。这使智能体能够承担更大型的任务，并以更高的自主性运行。截至撰写本文时，工业界或许最受欢迎的智能体是 Anthropic 的 Claude Code，Watanabe等人 [48]对其进行了审慎的描述。关于当代学术文献的概述，可参阅 He等人 [17] 和 Wang等人 [47] 的综述。

重构是编码智能体所研究的众多任务之一。例如，Xu 等人提出了 MANTRA，这是一种用于重构的多智能体框架 [49]。MANTRA 将三个智能体分别组织为 1）开发者、2）审查者和 3）修复角色，以执行重构任务，其表现优于直接使用大语言模型。然而，Claude 也是一个

能力出色的重构智能体，尽管它并非多智能体解决方案。在本研究中，我们以 Claude（v2.0.13）作为当代智能体人工智能重构的代表性示例。此外，截至本文撰写时，MANTRA 尚未公开发布。

## 3 方法

我们的目标是探究代码特征如何影响人工智能重构的能力，尤其关注代码健康度。我们提出了三个研究问题（RQ），从人工智能友好性的角度探究代码特征。首先，我们从大语言模型内在特性的角度出发，研究困惑度。这与相关工作相联系，并提供了一个基线。其次，我们考察代码健康度与下游人工智能任务成功率之间的关联。第三，我们检验代码健康度相较于源代码行数和困惑度的预测能力。

**研究问题**$_1$ 健康代码与不健康代码的困惑度有何差异？**研究问题**$_2$ 健康代码与不健康代码的人工智能重构失败率有何差异？**研究问题**$_3$ 代码健康度在多大程度上能够预测人工智能重构失败率？

### 3.1 数据集创建

我们从由 Google DeepMind 托管于 GitHub[2] 上的 CodeContests 数据集中进行采样。该数据集由 Li等人 [26] 首次提出，作为 AlphaCode 的训练数据，其中包含来自五个来源、针对竞赛编程问题的超过 1200 万个解答（正确和错误）。我们研究这一领域的代码，是因为这些问题配有精心设计的测试用例，可用于验证功能正确性，并在重构后提供实用的判定依据。

我们基于四项设计选择构建了一个包含 5,000 个解答的数据集。首先，我们决定专注于使用 Python 编写的解答，以提高新颖性和便利性。虽然 CodeContests 还包含 Java 和 C++编写的解答，但 Java 已在重构研究中得到广泛研究，而 C++ 具有更复杂的编译和测试流程。其次，我们要求解答中至少存在一个 CodeScene 代码异味。消除代码异味是一个现实的重构目标。第三，我们只研究包含 60 至 120 个源代码行数的解答。可维护性与规模之间存在很强的相关性 [41], ，因此我们在采样阶段就已经对此进行了控制，至少在一定程度上如此。第四，鉴于许多解答高度相似，我们主动寻求数据集的多样性。我们使用 CodeBleu [35] 进行相似度计算，采用以下权重：n元语法=0.1、加权n元语法=0.4、AST 匹配=0.5，以及数据流匹配=0。

实际抽样过程遵循以下步骤（括号内为剩余解的数量）：

(1) 下载完整的 CodeContests 数据集（13,210,440）。(2) 仅筛选 Python 3 解决方案（1,502,532）。(3) 删除所有完全相同的（类型1克隆）解决方案（1,390,531）。(4) 删除所有不含 CodeScene 代码异味的解决方案（88,156）。(5) 移除注释和无绑定的字符串字面量。(6) 筛选源代码行数为 60–120 的解决方案（18,074）。(7) 划分为两个层：健康与不健康代码[3]。(8) 对每个层重复以下步骤，直至纳入 2,500 个解决方案：(b) 运行相应的测试用例；若任何测试用例失败，则跳过。(c) 计算其与现有样本的 CodeBleu 相似度；若相似度 $\geq 0.9$，则跳过。(d) 将该解决方案加入此层。(9) 合并两个层，构成最终数据集（5,000）。

[2]https://github.com/google-deepmind/code_竞赛 [3]抽样脚本中的阈值设定存在一个小问题，导致代码健康度（CH） = 9 被判定为不健康。本文所有分析均采用预期的约定（健康： ≥ 9）。（a）随机抽取一个候选解。

### 3.2 大语言模型的选择

我们选择了五个参数量介于 20 至 30B、可在本地数据中心运行的开放权重模型。我们根据 Hugging Face 上的受欢迎程度和下载量统计数据选取了四个模型，并补充了一个由 IBM 最近发布的大语言模型——我们将这些称为中型大语言模型。此外，我们还纳入了一个最先进的大语言模型，并通过 Anthropic 的 API 向其发送提示。

**Gemma** gemma-3-27b-it（Google）– 2025年3月。**GLM** GLM-4-32B-0414（智谱AI）– 2025年4月。**Granite** Granite-4.0-H-Small（IBM）– 2025年10月。**GPT** gpt-oss-20b（OpenAI）– 2025年8月。**通义千问** Qwen3-Coder-30B-A3B-Instruct（阿里云）– 2025年8月。**Sonnet** claude-sonnet-4-5-20250929（Anthropic）– 2025年9月。

为在统一设置下实现重构多样性，我们将所有大语言模型的采样温度设为 0.7，并将生成上限设为 8,192 个新词元。其他设置均使用默认值。

### 3.3 RQ$_1$困惑度

我们提取了所研究的五个大语言模型在全部 5,000 个样本上的困惑度分数。随后，我们将样本分为健康与不健康两类，并针对每个大语言模型提出以下零假设：

$H_0$：健康代码与不健康代码的困惑度分布相同。

描述性统计显示，少量离群值的困惑度极高。人工分析发现了一些会显著增大大语言模型下一个词元不确定性的特定模式。在竞技编程中，一些参赛者会将天文数字般大的整数或很长的字符串硬编码，以便在后续解题时使用。为缓解这一问题，我们采用单侧稳健 z 分数过滤，并从大语言模型的输出中移除了 $z>2.5$ 的样本（约 5%）。

PPL 分数通常呈右偏分布，我们使用 Shapiro–Wilk 检验评估正态性。即使过滤掉上尾离群值后，我们仍拒绝所有大语言模型输出均服从正态分布的假设。随后，我们使用非参数曼-惠特尼 U 检验进行显著性检验。最后，我们使用 Holm 校正（Holm-Bonferroni 逐步下降法）对五个模型的多重比较进行调整（家族错误率 $\alpha$=0.05）。我们报告经过 Holm 校正的双侧 p值，并以 Cliff's $\delta$ 作为效应量。

### 3.4 RQ$_2$ 重构破坏率

我们要求所有中型大语言模型对这 5,000 个样本进行重构。对于词元价格较高的 Sonnet，我们对 1,000 个随机抽取的样本进行重构，其中健康和不健康样本数量相等。针对每种大语言模型与样本的组合，我们使用相同的提示词，遵循以下通用结构进行设计：

(A) 以角色描述的形式提供上下文，以引导模型 1
指向其知识中正确的部分。(B) 模型要执
行的一项具体任务。

(C) 关于模型应如何组织回答格式的说明。(D) 完成任务所需的输入数据。

重构提示

(A)请扮演一名资深软件工程师。(B)你的任务是重构以下 Python 代码，以提升其可维护性并使代码简洁。(C) 仅使用完整的重构后 Python 代码块作答。不要添加任何解释、注释或开场白。

(D)要重构的原始代码： ``` python <CODE> ```

我们使用 Claude（v2.0.13）补充了大语言模型的研究，以进一步考察一种用于重构的最先进智能体方法。我们之所以专门选择 Claude，是因为它被认为是工业界领先的模型，目前在公开的 SWE-bench 结果中名列前茅 [22]。Claude 在终端中以交互模式运行，但我们控制了环境，并以与大语言模型设置一致的方式呈现任务。Claude 是一项成本高昂的服务，因此我们决定选取与 Sonnet 相同的 1,000 个样本。我们的目标是将 Anthropic 解决方案的最终成本控制在 100 美元以下。

我们将随机样本批次组织在不同文件夹中，以模拟开发者可能的工作方式：先使用包含 20 个文件的小型样本进行试运行，随后使用 200、400 和 380 个文件的样本。我们将指令（A–C）写入公共父文件夹中的 CLAUDE.md 文件，并附加以下内容："此文件夹中的每个文件都是独立的；你不必担心文件之间的依赖关系。"

对于每个与 Claude 配合使用的样本文件夹，我们都添加了一个配置文件（.claude/settings.json），用于固定模型版本并限制代理的工具使用。具体而言，我们将模型设置为 claude-sonnet-4-5-20250929，明确禁用了 Bash、WebFetch 和 WebSearch，也禁用了任何 MCP 的使用。

对于每次重构会话，我们都遵循以下步骤：（1）在当前文件夹中启动 Claude。（2）对"你信任此文件夹中的文件吗？"回答"是"。（3）询问 Claude："你能看到父文件夹中的 CLAUDE.md 吗？"并授予读取权限。（4）给出指令："请重构此文件夹中的 Python 文件，以提高可维护性。"（5）当 Claude 请求编辑权限时，批准其在本次会话中编辑所有文件。

每次单独执行重构后，我们都会记录输出代码的描述性统计数据，计算其代码健康度，并运行相应的测试用例，以确定行为是否得到保留。如果任何测试用例失败，我们就将该重构称为失败。针对每种重构方法，我们提出以下零假设：

$H_0$：健康代码与不健康代码的重构失败率没有差异
健康与不健康代码 .

我们使用卡方（$\chi^2$）独立性检验比较破坏率，并报告风险差异（RD）和相对风险（RR）及其 95% 置信区间。按照 $RQ_1$ 的做法，我们使用 Holm 校正控制族系误差（$\alpha$=0.05）。

## 3.5 $RQ_3$ 代码健康度的预测能力

我们使用中型大语言模型重构输出的数据训练决策树，以探究哪些代码特征会影响重构破坏其测试套件的概率。我们的特征集旨在涵盖三个互补维度，同时避免冗余，具体考虑如下：

- **代码健康度**是我们关注的主要解释变量，体现了面向人类的可维护性。
- **困惑度**也被纳入考量，因为大语言模型的内部置信度可能包含预测信号。此外，$RQ_1$ 发现，困惑度与代码健康度相互正交（见第4节.2）。
- **源代码行数**被纳入作为一种简单的规模度量，通常可有效地作为结构复杂度的代理指标。Kotti 等人 [23]（以及我们针对$RQ_1$的佐证结果）发现，源代码行数与困惑度并不相关。
- **令牌数量**被排除在外，以尽量减少冗余，因为它与源代码行数相关（并且在$RQ_1$中显示出与困惑度之间虽小但一致的相关性）。

我们使用一棵固定的浅层决策树（最大深度为 = 3，每个叶节点的最小样本数为 = 25，并采用类别加权），以便优先保证可解释性，并便于比较六个大语言模型的结果。我们使用这些固定超参数进行5折交叉验证，然后在全部数据上重新拟合决策树，以便最终进行可视化并提取规则。最后，我们报告最终决策树的ROC曲线下面积（曲线下面积），这是一种与阈值无关、对类别不平衡具有稳健性的指标。不过请注意，我们训练决策树是为了进行解释，而非实现准确预测。

作为稳健性检验，我们使用相同的数据和特征拟合了逻辑回归模型。逻辑回归会给出优势比（OR），表示在其他条件不变时，某个预测特征每增加一个标准差，测试通过的优势会发生怎样的变化。

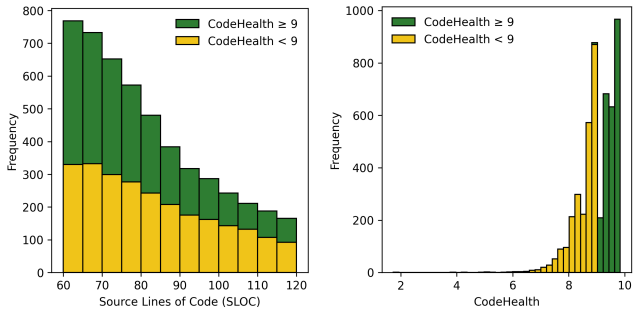
# 4 结果

本节首先介绍数据集的描述性统计，随后给出三个研究问题的结果。所有相关的 Jupyter笔记本均可在复现包 [7] 中获取。

## 4.1 描述性统计

图1展示了数据集中 5,000 个解决方案的源代码行数和代码健康度分布。对于源代码行数，解决方案集中在下界附近，随后随着规模增大而稳步减少。对于代码健康度，分布呈左偏态，并且可以观察到分层抽样的影响。在两个层中，代码健康度较高的解决方案密度都较大，这解释了代码健康度 = 9 以上不远处出现的低谷。总体而言，代码健康度的分布与先前研究 [9,44],报告的模式相似，不过本文研究的短篇竞赛编程任务中，这种偏态不那么明显。

CodeScene 在数据集中识别出九种不同的代码异味。最常见的五种异味（及其数量）是：1）**崎岖之路** (n=4,901)。一个函数包含多个嵌套控制结构片段，这表明缺少抽象，使代码更难理解。2）**复杂方法** (n=3,572)。一个函数的圈复杂度较高，意味着它包含许多独立的逻辑执行路径。3）**深层嵌套复杂度**] (n=2,433)。循环和条件语句等控制结构彼此嵌套，层级众多。4）**复杂**




**(a) 源代码行数分布。** **(b) 代码健康度分布。**

**图1：源代码行数和代码健康度的分布。**

**条件语句** (n=1,328)。某些表达式在单个条件中组合了多个逻辑运算，例如合取和析取。5）**过多的函数参数** (n=724)。一个函数接受的参数过多，这表明该函数承担了过多职责，或缺少恰当的抽象。

## 4.2 RQ$_1$困惑度

表 1 汇总了各中型大语言模型按代码健康度分组划分的困惑度。我们发现，除 Granite 外，所有模型的困惑度分布均存在显著差异。然而，不同模型的差异方向不一，且所有模型的效应量都微乎其微。总体而言，本研究得出了否定性结果，即我们未发现困惑度与代码健康度之间存在具有实际意义的关联。

我们重新审视了 Kotti 等人 [23],得出的否定性发现：他们报告文件大小与困惑度之间不存在相关性。我们使用 Spearman 相关系数（$\rho$）分析筛选后的数据集，基本证实了这一结论：对于源代码行数，除 GPT 显示出较弱的正相关（$\rho = +0.197$）外，所有模型的关联都微乎其微。

我们还考察了令牌数量，其中四个模型呈现较弱的负相关：Gemma（$\rho = -0.197$）、GLM（$\rho = -0.242$）、$\rho = -0.223$）、$\rho = -0.245$）、Granite（），以及通义千问（）。同样，GPT 表现突出，这次呈现较弱的正相关（$\rho = +0.153$）。总体而言，随着令牌数量增加，大语言模型对代码的困惑度往往略有下降，而源代码行数则没有呈现相应的规律。

**研究问题1：健康代码与不健康代码的困惑度有何差异?**

不同模型的结果方向不一，且效应量微乎其微，这表明 PPL 与代码健康度之间不存在具有实际意义的关联。

## 4.3 研究问题$_2$ 重构破坏率

表2列出了各重构方法的测试判定，并按健康代码和不健康代码分类。大语言模型按破坏率从低到高排序，智能体方案 Claude 则列在双横线下方。

结果显示，所有中型大语言模型处理健康代码时的破坏率都始终较低。风险差异均具有统计显著性，其中通义千问为 -8.58 个百分点，Gemma 为 -15.12 个百分点。相对风险从通义千问的 0.692 到 Granite 的 0.852 不等。这意味着，在本研究中能力最强的中型大语言模型，其风险降低幅度超过 30%。对于最先进的大语言模型 Sonnet，健康代码与不健康代码之间没有显著差异。Claude 的风险表现则明显更为保守。在第一个重构任务中，无论代码是健康还是不健康，Claude 破坏的测试套件都只有约 5%——同样，这一差异在统计上并不显著。

为了进一步考察各重构方法在执行任务时的成效，我们分析了它们对代码健康度的影响。表3展示了结果，排序方式与表2相同。我们报告通过测试套件的重构输出中的代码健康度变化，并将其分为增加（代码健康度 ↑）、无变化（代码健康度 ↔）和降低（代码健康度 ↓）。最后一列（成功率）报告成功重构的比例，即代码健康度得到改善且测试通过。

不同大语言模型在保持行为不变的重构中，代码健康度变化差异显著。所有大语言模型有时都会降低代码健康度，这意味着要么 1）新增了代码异味，要么 2）现有代码异味的严重程度有所增加。我们还观察到健康代码与不健康代码之间存在一致的差异：当初始代码不健康时，大语言模型更常提高代码健康度。最大差异出现在 Sonnet 和 GLM 之间（约 25 个百分点），最小差异则出现在 Granite（约 13 个百分点）。对于健康代码和不健康代码，Sonnet 的代码健康度改善率分别为 57.74% 和 82.42%，优于中型大语言模型。

对于中型大语言模型，结果揭示了行为保持与改善代码健康度之间的权衡。在这些模型中，通义千问最能保持行为不变，但改善代码健康度的频率也较低，尤其是在健康代码上，代码健康度通常没有变化（46.28%），仅有 26.71% 的情况有所改善。其他所有中型大语言模型提高代码健康度的比例都更高，无论是针对健康代码还是不健康代码。观察最后一列可以发现，在中型大语言模型中，GPT 成功重构的比例最高（分别为 38.44% 和 43.52%），其次是 GLM 和通义千问。结果展现了大语言模型在更大胆地尝试重构与保持行为之间的平衡。作为对比，Sonnet 对健康代码和不健康代码的成功重构比例分别为 50.10% 和 69.26%。

Claude 的结果表明，在基于大语言模型构建智能体产品时，如何平衡这种权衡。我们发现，在所研究的所有重构方法中，Claude 最为保守：健康代码和不健康代码的代码健康度分别有 68.81% 和 60.76% 的情况保持不变。这也与 Claude 在完成一批重构操作后通常提供的输出摘要一致，摘要描述的变更主要包括：1）重命名变量，使其符合代码意图；2）按照约定组织代码；3）调整空白格式（摘要见附录 B）。我们手动分析了一部分重构，以确认这一行为。由于这些操作很少影响 CodeScene 的代码异味，因此通常不会改变代码健康度，这也解释了为何它往往保持不变。

然而，Claude 有时会尝试更大胆的重构，导致代码改动更具侵入性。附录 B 中的第二个摘要展示了这样一个例子。这段引文列出了若干已完成的重构操作，它们有可能提高代码健康度，

**表 1：剔除离群值后，五种大语言模型各代码健康度组的困惑度。** $\Delta\tilde{x}$ **= 健康 − 不健康。p值经 Holm 校正；** $\delta$ **为 Cliff效应量。**

| 模型 | 组 | $N$ | 中位数 | IQR | Max | $\Delta\tilde{x}$ | $p$ | $\delta$ |
|---|---|---|---|---|---|---|---|---|
| **Gemma** | 健康 | 2369 | 2.092 | 0.515 | 3.127 | +0.069 | <0.001 | +0.098 |
| | 不健康 | 2396 | 2.023 | 0.552 | 3.071 | | | |
| **GLM** | 健康 | 2375 | 1.577 | 0.493 | 2.529 | -0.017 | 0.039 | -0.039 |
| | 不健康 | 2404 | 1.594 | 0.492 | 2.533 | | | |
| **GPT** | 健康 | 2214 | 2.943 | 1.473 | 6.226 | +0.237 | <0.001 | +0.136 |
| | 不健康 | 2216 | 2.706 | 1.559 | 6.299 | | | |
| **Granite** | 健康 | 2369 | 1.764 | 0.582 | 2.932 | +0.005 | 0.992 | +0.000 |
| | 不健康 | 2395 | 1.759 | 0.617 | 2.923 | | | |
| **Qwen** | 健康 | 2335 | 1.728 | 0.610 | 2.915 | -0.041 | 0.004 | -0.054 |
| | 不健康 | 2388 | 1.769 | 0.622 | 2.968 | | | |

**表2：各重构方法的测试判定，以及** $\chi^2$ **健康与不健康代码破坏率的比较。大语言模型按破坏率从低到高排列。p值经过 Holm 校正。RD 表示以百分点计的风险差异，RR 表示相对风险（健康代码/不健康代码）。方括号内为 95%置信区间。**

| | | 总计 | 通过的测试 | | $\chi^2$ (健康 vs. 不健康) | | |
|---|---|---|---|---|---|---|---|
| 模型 | 组 | | N | % | p | 风险差（百分点） [95% 置信区间] | 相对风险 [95% 置信区间] |
| **Sonnet** | 健康 | 499 | 433 | 86.77 | 0.439 | −2.74 [−7.11, 1.63] | 0.828 [0.613, 1.120] |
| | 不健康 | 501 | 421 | 84.03 | | | |
| **Qwen** | 健康 | 2501 | 2019 | 80.72 | <0.001 | −8.58 [−10.92, −6.24] | 0.692 [0.625, 0.766] |
| | 不健康 | 2499 | 1803 | 72.16 | | | |
| **GPT** | 健康 | 2501 | 1604 | 64.13 | <0.001 | −11.15 [−13.87, −8.44] | 0.763 [0.713, 0.816] |
| | 不健康 | 2499 | 1324 | 52.98 | | | |
| **GLM** | 健康 | 2501 | 1504 | 60.14 | <0.001 | −10.16 [−12.90, −7.41] | 0.797 [0.749, 0.848] |
| | 不健康 | 2499 | 1249 | 50.02 | | | |
| **Gemma** | 健康 | 2501 | 1394 | 55.74 | <0.001 | −15.12 [−17.86, −12.38] | 0.745 [0.706, 0.787] |
| | 不健康 | 2499 | 1015 | 40.58 | | | |
| **Granite** | 健康 | 2501 | 1162 | 46.46 | <0.001 | −9.29 [−12.01, −6.56] | 0.852 [0.813, 0.894] |
| | 不健康 | 2499 | 929 | 37.17 | | | |
| **Claude** | 健康 | 499 | 480 | 96.19 | 0.439 | −1.38 [−3.95, 1.19] | 0.734 [0.411, 1.308] |
| | 不健康 | 501 | 475 | 94.81 | | | |

包括：函数提取、简化复杂条件、消除代码重复以及扁平化嵌套结构。我们没有发现 Claude 选择保守或更大胆的重构尝试时存在任何规律。同时，我们

请注意，Claude 关于“零功能变更”和“文件保持其原始功能”的表述过于自信——有时该代理确实会破坏行为。

**表3：非破坏性重构的测试判定和代码健康度变化。占比最大的部分以粗体显示。"成功率"是整个队列中代码健康度提高（即代码异味消除）且通过测试的部分（CH ↑/总计）。**

| | | 总计 | 通过的测试 | | 通过的测试中的代码健康度变化 | | | | | | 成功率 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 模型 | 组 | | N | % | CH ↑ | % | CH ↔ | % | CH ↓ | % | |
| **Sonnet** | 健康 | 499 | 433 | 86.77 | 250 | **57.74** | 69 | 15.94 | 114 | 26.33 | 50.10 |
| | 不健康 | 501 | 421 | 84.03 | 347 | **82.42** | 30 | 7.13 | 44 | 10.45 | 69.26 |
| **Qwen** | 健康 | 2501 | 2019 | 80.72 | 539 | 26.71 | 934 | **46.28** | 545 | 27.01 | 21.56 |
| | 不健康 | 2499 | 1803 | 72.16 | 853 | **47.28** | 581 | 32.21 | 370 | 20.51 | 34.12 |
| **GPT** | 健康 | 2501 | 1604 | 64.12 | 961 | **59.95** | 305 | 19.03 | 337 | 21.02 | 38.44 |
| | 不健康 | 2499 | 1324 | 53.00 | 1088 | **82.11** | 107 | 8.08 | 127 | 9.58 | 43.52 |
| **GLM** | 健康 | 2501 | 1504 | 60.12 | 679 | **45.18** | 414 | 27.54 | 410 | 27.29 | 27.16 |
| | 不健康 | 2499 | 1249 | 50.00 | 880 | **70.40** | 158 | 12.64 | 212 | 16.96 | 35.20 |
| **Gemma** | 健康 | 2501 | 1394 | 55.76 | 627 | **44.98** | 426 | 30.56 | 341 | 24.46 | 25.08 |
| | 不健康 | 2499 | 1015 | 40.60 | 685 | **67.49** | 162 | 15.96 | 168 | 16.55 | 27.40 |
| **Granite** | 健康 | 2501 | 1162 | 46.44 | 417 | 35.90 | 540 | **46.52** | 204 | 17.58 | 16.68 |
| | 不健康 | 2499 | 929 | 37.20 | 454 | **48.82** | 347 | 37.31 | 129 | 13.87 | 18.16 |
| **Claude** | 健康 | 499 | 480 | 96.19 | 96 | 20.00 | 331 | **68.96** | 53 | 11.04 | 19.24 |
| | 不健康 | 501 | 475 | 94.81 | 124 | 24.75 | 288 | **60.63** | 63 | 13.26 | 24.75 |

**表4：使用大语言模型生成的 5,000 个输出重构训练时的决策树结果。最重要特征以粗体显示。**

| LLM | %Break | AUC | 特征重要性 | | |
|---|---|---|---|---|---|
| | | | 代码健康度 | 困惑度 | SLOC |
| Qwen | 0.236 | 0.553 | **0.707** | 0.160 | 0.132 |
| GPT | 0.414 | 0.559 | **0.683** | 0.268 | 0.049 |
| GLM | 0.449 | 0.546 | **0.572** | 0.360 | 0.068 |
| Gemma | 0.518 | 0.565 | **0.880** | 0.120 | <0.001 |
| Granite | 0.582 | 0.544 | **0.583** | 0.417 | <0.001 |

**研究问题2：人工智能重构的失败率在健康代码与不健康代码之间有何差异?**

五个中等规模的大语言模型在健康代码上的破坏率显著更低，最弱模型的相对风险降低约为 15%，最强模型则约为 30%。Sonnet 的结果不太明确；而当其作为代理式 Claude 配置中的大语言模型使用时，解决方案始终较为保守（破坏率为 ≈5%）。

## 4.4 研究问题$_3$ 代码健康度的预测能力

表4展示了使用中型大语言模型的重构数据训练解释性决策树所得的结果。"%Break"表示破坏率，曲线下面积（AUC）表示拟合后的决策树区分通过测试与失败测试的能力。最后三列展示了所研究的三个特征的相对重要性：代码健康度、困惑度和源代码行数。

尽管所有拟合出的决策树都不擅长对失败的测试进行分类，但代码健康度始终是区分度最高的特征。它在全部五棵树中都作为根节点分裂特征出现，下面列出了针对各大语言模型的顶层决策阈值。括号中的数字表示仅遵循这一条规则时达到的朴素分类准确率，即被正确分类为失败的样本占比。对于其中四个大语言模型，学得的阈值接近 CodeScene 对健康代码设定的默认边界（$CH = 9$）；此前研究表明，该边界与人类可维护性判断相符 [6]。

**通义千问** 代码健康度 ≤ 8.895（63% – 1163 次失败，848 次通过）
**GPT** 代码健康度 ≤ 8.775 （ 61% – 1070 次失败，683 次通过）
**GLM** 代码健康度 ≤ 9.195（55% – 1495 次失败，1230 次通过） **通义千问** 代码健康度 ≤ 8.875（62% – 866 次失败，572 次通过）
**Granite** 代码健康度 ≤ 8.285（60% – 398 次失败，234 次通过）

困惑度是所有决策树中的第二重要特征。不过，它相对于源代码行数的重要性在不同大语言模型之间差异显著。对于通义千问，困惑度和源代码行数的重要性大致相当。另一方面，对于 Granite 和 Gemma，源代码行数完全不具备预测信号。

图 2 展示了通义千问的完整决策树，作为一个代表性示例。该树说明代码健康度如何构成根节点决策，后续则根据困惑度和源代码行数进行划分，或再次依据代码健康度划分。我们用 fail 表示会造成破坏的重构，用 pass 表示不会造成破坏的重构。大多数失败的重构集中在代码健康度较低的分支中，这支持了上述整体趋势。

作为稳健性检验，我们使用与决策树相同的数据训练了逻辑回归模型。我们再次采用 5 折交叉验证评估模型，以便在所有可用数据上拟合模型，所得曲线下面积分数与决策树相当。更重要的是，代码健康度的优势比为：通义千问 1.347、GPT 1.307、GLM 1.215、Gemma 1.420 和 Granite 1.240。这意味着，代码健康度每增加一个标准差（在该数据集中约为 0.65），成功（未导致破坏）的重构的优势就会提高约 20%–40%，这证实了在使用中型大语言模型进行人工智能重构时，更健康的代码不太容易被破坏。

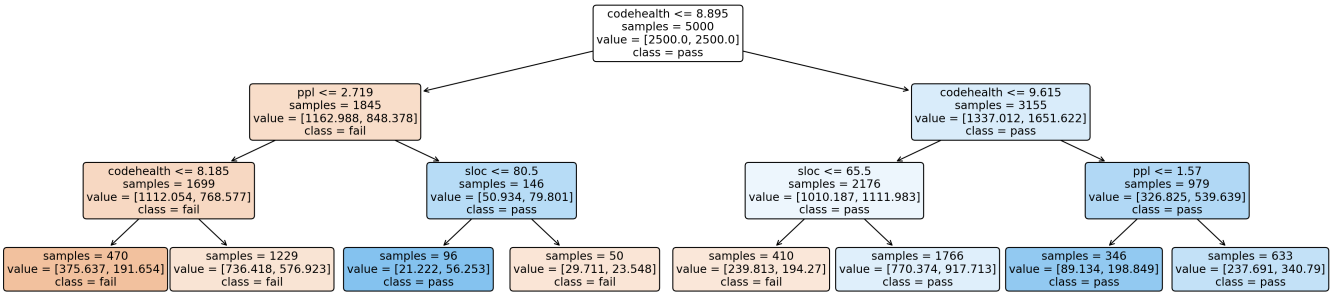



**图 2：通义千问的决策树。节点显示：1）分裂规则（首行；叶节点中省略）；2）到达该节点的样本数；3）[失败、通过], 的类别加权计数；4）预测的多数类，也通过颜色表示。**

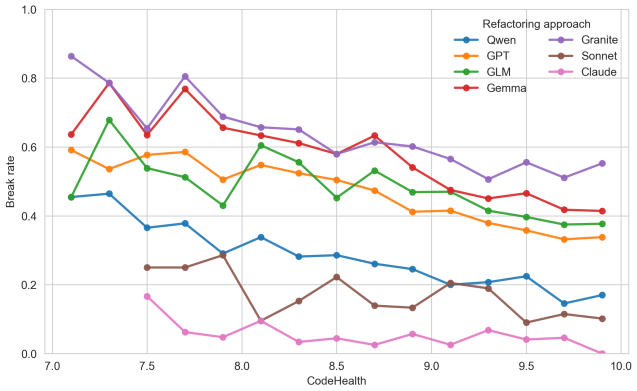



**图 3：按 0.2 为间隔分箱后，重构失败率随代码健康度的变化。为减少噪声，省略样本数少于 10 的箱。**

图 3 展示了重构失败率随代码健康度变化的情况。所有大语言模型都呈现出明显的负向趋势，即针对更健康代码进行的重构较少导致测试失败。我们注意到，即使在 $RQ_2$ 中，Sonnet 的健康与不健康代码之间的差异并不显著，这一趋势仍然成立。相反，我们的结果表明，能力更强的大语言模型会将阈值移至更低的值，也就是说，Sonnet 的安全区间可能从代码健康度≈ 8左右开始。然而，对于智能体方案 Claude（本研究考察的最保守重构方法），结果并未显示出明显趋势。

**研究问题 3：代码健康度在多大程度上能够预测人工智能重构失败率?**

代码健康度为重构成功提供了适度但一致的信号，其预测信息比困惑度或源代码行数多出 ≈3–10× 。面向人类校准的"健康"阈值（代码健康度=9）也划分出了人工智能重构中较低的破坏风险。

## 5 讨论

本节结合先前研究讨论我们的发现、其影响以及最重要的效度威胁。

### 5.1 综合与解读

首先，我们发现代码健康度与困惑度在很大程度上彼此正交。先前研究表明，代码健康度与人类对可维护性的感知高度一致 [6], ，而这一发现与先前研究提出的人类代码理解能力与困惑度之间存在关联的观点相悖 [1, 12]。仅依赖困惑度来评估代码是否易于理解，似乎过于天真。

其次，我们发现健康代码与不健康代码的破坏率存在差异。在包含 5,000 个样本的较大规模实验中，所有中型大语言模型都表现出显著且具有实际意义的风险差异。在 Anthropic 最先进的方法所进行的较小规模实验中（1,000 个样本），我们观察到了相同的趋势，但效应较小，且未达到统计显著性。尽管如此，较高的代码健康度意味着更好的结构，这似乎有助于大语言模型保留语义——这一发现与 Thoughtworks 早期关于"*AI* 友好型代码设计"的观察一致 [43]。最后，Claude 的表现比直接使用大语言模型更为保守，这很可能反映了该公司有意作出的产品决策。

第三，随着代码健康度提高，我们观察到重构失败率持续下降。这有力表明，人类开发者和大语言模型在代码充斥着 CodeScene 的代码异味时都会面临更大困难。尽管我们没有发现困惑度与之存在有意义的关联，但我们的结果支持 Abdelsalam 等人的总体观点，即"人类和大语言模型同样容易感到困惑"。最后，与困惑度和简单的规模指标源代码行数相比，代码健康度对重构破坏风险具有更强的预测能力。

### 5.2 对实践的启示

规划如何采用人工智能辅助软件开发，如今已成为大多数软件组织的优先事项。每个组织都必须决定如何在持续进行的人工智能转型中定位自身，并评估其竞争影响。先前研究建议采取受控推出并进行谨慎治理 [15],强调让人类参与其中 [3, 36], [5, 呼吁重18], [28] 视人工代码审查流程，并指出需要高质量的测试套件 。

我们的研究结果表明，考虑CH的部署策略可以成为明智采用人工智能的一部分。与人类一样，大语言模型在代码健康度较高的代码上表现更好。因此，我们建议：1）尽早将人工智能部署重点放在健康且"适合AI"的代码上；或 2）在扩大大语言模型驱动的解决方案之前，明确提高代码库目标区域的代码健康度。

以人为参与环节的大语言模型驱动解决方案。以往研究强调人在环路中的实践，而我们的研究补充了一点：代码健康度越低，预期需要越多的人工参与。凭借当前的人工智能能力，在最棘手的旧遗留部分中，无法期待快速取得进展。

较高的代码健康度可以作为人工智能部署的软性门槛，但具体阈值必须视任务而定，并取决于大语言模型。在我们的重构场景中，CH= 9 对于中型大语言模型而言是一个合理的参考值，而能力更强的重构方法则可能能够处理代码健康度更低、更具挑战性的代码。展望未来，如何在大语言模型成本与任务复杂度之间找到最有效的平衡，将是一项重要的商业决策，这与 Zup Innovation 在定制编码智能体 [32] 时总结的第一条经验一致。

对于较为保守的人工智能解决方案，例如让 Claude 完成第一轮重构，预期主要会出现格式化和命名更改。虽然这些更改不会改善代码健康度，但仍然有价值，因为它们或许能够拨开迷雾，帮助人们发现重点。先前研究表明，如果变量名称恰当，开发者能更快地在代码中发现缺陷 [20]。

最后，先前研究报告指出，代码健康度越高，价值收益的提升越显著 [9]。我们预计，代码智能体的广泛采用将加剧这一效应。我们已经知道，人类能够更快地演进健康代码；我们的研究结果表明，机器也同样如此。随着代码智能体日益普及，若不维护健康代码库，人工智能加速的竞争对手就会凭借更快的速度超越你——这与我们在敏捷转型期间的经历相似 [10]。

## 5.3 对研究的启示

我们的结果为研究基于测量的人工智能辅助开发可行性（即人工智能友好性）开辟了新方向。一个有前景的方向是，找到更好的先验估计方法，以判断人工智能解决方案可能在哪些代码库中取得成功。

在文件级，PPL 并不是衡量人工智能友好性的良好指标。不过，我们发现人类和机器至少在一定程度上会受到相同代码异味的困扰，而 Abdelsalam等人报告称，词元级意外度与人类困惑之间存在关联 [1]。这表明，在更细的粒度上进行研究具有潜力。未来工作应探讨如何更好地将 PPL 用于评估人工智能友好性，同时避免采用会掩盖局部困惑的文件级平均。

在文件级，代码健康度更能反映人工智能友好性。不过，仍需研究如何将代码健康度与反映结构可编辑性和行为保持能力的其他方面结合起来。可考虑的指标包括跨文件依赖度量、类型安全覆盖率和副作用范围。

学术界和工业界对多智能体解决方案的兴趣日益浓厚。我们认为，将编码智能体与代码健康度信息相结合，潜力巨大；既可以通过独立的代码质量智能体实现，也可以将其作为编码智能体工具箱中的工具，类似于运行测试或代码检查。模型上下文协议（MCP）目前是一种流行的集成方式，但仍需进一步研究，以了解它对这些不断发展的生态系统中的可维护性和安全性的影响 [16]。

## 5.4 效度威胁

本小节将按照重要性顺序讨论主要的效度威胁。首先，我们介绍代码级 AI 友好性这一构念。接下来有两个问题 [40]：1）这一概念是否有效？以及 2）我们的测量是否足够全面？我们认为，代码中存在某些模式，会让大语言模型更容易或更难对其进行修改，因此这一概念是有效的。然而，将人工智能重构的结果用作该构念的替代指标存在局限。对大语言模型而言，其他任务可能更容易，例如测试用例生成、文档编写以及解释代码意图。我们对 AI 就绪度的测量针对的是侵入性代码更改，因此在其他开发任务中，结果可能有所不同。

其次，我们的数据集由竞赛性编程中的 Python 解答构成。这一小众领域侧重于快速编写功能正确的代码 [24]，，可维护性几乎无关紧要。我们见过一些绝不会进入生产代码库的代码片段。此外，这些代码具有算法性，与大多数开发者的日常工作相去甚远。尽管如此，我们认为，研究人工智能重构在这类代码上的表现，是考察人工智能友好性的一个有意义的视角。未来的工作应研究多种语言中的其他类型代码。

最后，我们报告了内部效度和结论效度方面的威胁，但认为这些威胁较小。首先，由于大语言模型具有非确定性，因此存在运行间方差。大量样本足以缓解这一威胁，因此我们没有采用重复尝试或 pass@k 指标。其次，我们针对中型大语言模型（N=5,000）和最先进的解决方案（N=1,000）开展的实验运行存在统计功效不对称。这可能导致一些实际存在但统计功效不足的效应未达到显著水平。

## 6 结论与未来工作

未来的代码库必须服务于由人类开发者和编码智能体组成的混合型团队。数十年来，程序理解研究为我们提供了如何编写符合人类大脑特点的代码的指导。近期，Thoughtworks 创造了“AI 友好型代码设计”这一术语。在本研究中，我们探讨面向人类设计的编码实践是否也有助于机器。我们通过代码健康度来开展研究：这是一种根据人类认知进行校准的可维护性指标；我们还研究了它与人工智能重构成功率之间的关系。

我们的结果证实，人类友好型代码更适合接受人工智能干预。代码健康度越高，人工智能重构越能保留程序语义。在中型大语言模型中，健康代码的破坏率显著更低。最先进的大语言模型 Sonnet-4.5 也呈现相同趋势，但其破坏率整体更低。Claude Code 通常较为保守，即优先考虑正确性而非收益，且未显示出健康与不健康代码之间的显著差异。

除了组织和流程因素外，代码质量也应纳入 AI 辅助开发的部署决策。健康代码表明代码库中更安全的起点，而不健康代码则需要更严格的防护措施和更多人工监督。在实践中，代码质量很可能是实现 AI 加速所承诺效益的先决条件。

作为未来工作，我们将扩大实证研究。首先，我们将不再局限于竞赛编程，并纳入更多编程语言。我们尤其关注能够更好反映工业软件工程的生产代码。其次，我们将研究代码质量非常差时的重构失败率，因为我们的数据集中代码健康度为 $< 7$ 的样本过少。无论是专有项目还是开源项目中，都确实存在质量更差的代码。最后，由于大语言模型不断推出新版本，我们将重新评估这些新版本，以检验所观察到的趋势是否仍然存在。

## 致谢

本项工作部分由瓦伦堡人工智能、自主系统与软件计划（WASP）资助，部分由VINNOVA先进数字化计划资助的NextG2Com能力中心资助，资助编号为2023-00541。

## 参考文献

[1] Youssef Abdelsalam、Norman Peitek、Anna-Maria Maurer、Mariya Toneva 和 Sven Apel。2025。《人类和大语言模型如何处理令人困惑的代码？》 doi:10. 48550/arXiv.2508.18547[2] Harold Abelson 和 Gerald Sussman。1984。《计算机程序的构造和解释》。麻省理工学院出版社。 [3] Leonardo Banh、Florian Holldack 和 Gero Strobel。2025。副驾驶式协作的未来：生成式人工智能如何改变软件工程。《信息与软件技术》183 (2025)，107751。 doi:10.1016/j.infsof.2025.107751[4] Fraol Batole、Abhiram Bellur、Malinda Dilhara、Mohammed Raihan Ullah、Yaroslav Zharov、Timofey Bryksin、Kai Ishikawa、Haifeng Chen、Masaharu Morimoto、Shota Motoura、Takeo Hosomi、Tien N. Nguyen、Hridesh Rajan、Nikolaos Tsantalis 和 Danny Dig。2025。利用大语言模型、集成开发环境和语义嵌入自动执行移动方法重构。doi: 10 48550 2503 20934 。/arXiv。 。

[5] Stefano Bistarelli、Marco Fiore、Ivan Mercanti 和 Marina Mongiello。2025。。将大语言模型用于代码生成任务：综述。《*SN*计算机科学》 6, 6 2025 673 10 1007 42979 025 04241 5 ( )，。doi: 。/s - - - Markus

[6] Borg、Marwa Ezzouhri 和 Adam Tornhill。2024。揭示人工智能的回声：基于人工评估对可维护性指标和机器学习预测进行基准测试。载于第*40*届国际软件维护与演化会议论文集。678–688。 [7] Markus Borg 和 Nadim Hagatulah。2026。AI友好性复现包。https://github.com/codescene-research/ai-friendliness-forge-2026[8] Markus Borg、Dave Hewett、Nadim Hagatulah、Noric Couderc、Emma Söderberg、Donald Graham、Uttam Kini 和 Dave Farley。2025。人工智能的回声：探究人工智能助手对软件可维护性的下游影响。doi:10. 48550/arXiv.2507.00788[9] Markus Borg、Ilyana Pruvost、Enys Mones 和 Adam Tornhill。2024。持续提升，而非逐渐减弱：探究高可维护性代码的回报。载于第7届国际技术债务会议论文集。21–30。 [10] Jan Bosch。2017。速度、数据与生态系统：在软件驱动的世界中脱颖而出。博卡拉顿。[11] Jürgen Börstler、Kwabena E. Bennin、Sara Hooshangi、Johan Jeuring、Hieke Keuning、Carsten Kleiner、Bonnie MacKellar、Rodrigo Duran、Harald Störrle、Daniel Toll 和 Jelle van Assema。2023。开发者谈代码质量。《实证软件工程》 28, 6 2023 128 10 1007 10664 023 10381 0 ( )，。

[12] doi: 。/s - - - Casey Casalnuovo、Prem Devanbu 和 Emily Morgan。2020。。意外度能否预测代码理解难度？认知科学学会年会论文集42，0 (2020)。 [13] Hugging Face。2025。定长模型的困惑度。 https://huggingface.co/docs/transformers/en/perplexity[14] Dan Gopstein、Jake Iannacone、Yu Yan、Lois DeLong、Yanyan Zhuang、Martin K.-C. Yeh 和 Justin Cappos。2017。理解源代码中的误解。载于第*11*届软件工程基础联合会议论文集。计算机协会，美国纽约州纽约市，129–139。doi:10. 1145/3106237.3106264[15] Kaze m Haki、Dorsa Safaei、Adolfo Magan 和 Martin Griffiths。2025。将生成式人工智能融入企业平台：来自 Salesforce 的见解。《信息系统期刊》35，5 (2025)，1497–1512。doi:10.1111/isj.12593[16] Mohammed Mehedi Hasan、Hao Li、Emad Fallahzadeh、Gopi Krishnan Rajbahadur、Bram Adams 和 Ahmed E. Hassan。2025。初探模型上下文协议 (MCP)：研究 MCP服务器的安全性与可维护性。doi:10. 48550/arXiv.2506.13538[17] Junda He、Christoph Treude 和 David Lo。2025。基于大语言模型的多智能体软件工程系统：文献综述、愿景与未来发展。《*ACM*软件工程方法论汇刊》 34, 5 2025 124 1 124 30 10 1145 3712003 ( )，：–：。

[18] doi: 。/Lo Heander、Emma Söderberg 和 Christofer Rydenfält。2025。。支持而非自动化：迈向以人工智能辅助代码审查，提升代码质量及其他方面。载于第*33*届*ACM*软件工程基础国际会议论文集。计算机协会，591–595。 [19] Abram Hindle、Earl T. Barr、Mark Gabel、Zhendong Su 和 Premkumar Devanbu。2016。论软件的自然性。《*ACM*通讯》59，5 (2016)，122–131。doi:10.1145/2902362[20] Johannes C. Hofmeister、Janet Siegmund 和 Daniel V. Holt。2019。较短的标识符名称需要更长时间才能理解。《实证软件工程》24，1 (2019年2月)，417–443。 doi:10.1007/s10664-018-9621-x [21] F. Jelinek、R. L. Mercer、L. R. Bahl 和 J. K. Baker。1977。困惑度：衡量语音识别任务难度的指标。《声学学会期刊》 *of America* 62，1 1977 63 10 1121 1 2016299 S（），S。doi: 。/。

[22] Carlos E. Jimenez、John Yang、Alexander Wettig、Shunyu Yao、Kexin Pei、Ofir Press 和 Karthik R. Narasimhan。2024。SWE-bench：语言模型能否解决真实世界的 GitHub 问题？收录于 第*12*届国际学习表征会议论文集。[23] Zoe Kotti、Konstantina Dritsa、Diomidis Spinellis 和 Panos Louridas。2025。傻瓜确信不疑，智者满腹狐疑：探索代码补全中的大语言模型置信度。doi:10.48550/arXiv.2508.16131 arXiv:2508.16131 [cs]。 [24] Antti Laaksonen。2024。《竞赛编程指南：通过竞赛学习和改进算法。施普林格国际出版公司，查姆。[25] Guilherme Lacerda、Fabio Petrillo、Marcelo Pimenta 和 Yann Gael Gueheneuc。2020。代码异味与重构：挑战与观察的三级系统综述。《系统与软件杂志》167（2020），110610。[26] Yujia Li、David Choi、Junyoung Chung、Nate Kushman、Julian Schrittwieser、Rémi Leblond、Tom Eccles、James Keeling、Felix Gimeno、Agustin Dal Lago、Thomas Hubert、Peter Choy、Cyprien de Masson d' Autume、Igor Babuschkin、Xinyun Chen、Po-Sen Huang、Johannes Welbl、Sven Gowal、Alexey Cherepanov、James Molloy、Daniel J. Mankowitz、Esme Sutherland Robson、Pushmeet Kohli、Nando de Freitas、Koray Kavukcuoglu 和 Oriol Vinyals。2022。使用 AlphaCode 进行竞赛级代码生成。《科学》378，6624（2022），1092–1097。doi: 10 1126 1158 。/science.abq Antonio Martini、

[27] Terese Besker 和 Jan Bosch。2018。技术债务跟踪：实践现状——对15家大型组织的调查与多案例研究。《计算机编程科学》163（2018），42–61。[28] Noble Saji Mathews 和 Meiyappan Nagappan。2024。测试驱动开发与基于大语言模型的代码生成。收录于 第*39*届国际自动化软件工程会议论文集。 1583 1594 10 1145 3691620 3695527 –。doi: 。/。

[29] James Meaden、Michal Jarosz、Piotr Jodłowski 和 Grigori Melnik。2025。。COMPASS：用于评估大语言模型代码生成的多维基准测试。doi: 10 48550 2508 13757 。/arXiv。。Marvin Munoz Baron、

[30] Marvin Wyrich 和 Stefan Wagner。2020。认知复杂度作为源代码可理解性度量的实证验证。收录于第*14*届国际实证软件工程与度量研讨会论文集。1–12。 doi:10.1145/3382494.3410636[31] Stack Overflow。2025。*2025* 年开发者调查。技术报告。 https://survey.stackoverflow.co/2025/[32] Gustavo Pinto、Cleidson De Souza、Joao Batista Neto、Alberto Souza、Tarcisio Gotto 和 Edward Monteiro。2024。构建 StackSpot AI 的经验：情境化人工智能编程助手。收录于 第*46*届国际软件工程会议：软件工程实践论文集。408–417。doi:10.1145/3639477.3639751[33] Dorin Pomian、Abhiram Bellur、Malinda Dilhara、Zarina Kurbatova、Egor Bogomolov、Timofey Bryksin 和 Danny Dig。2024。新一代重构：结合大语言模型洞见与集成开发环境功能来提取方法。收录于 第*40*届国际软件维护与演化会议论文集。[34] Baishakhi Ray、Vincent Hellendoorn、Saheel Godhane、Zhaopeng Tu、Alberto Bacchelli 和 Premkumar Devanbu。2016。论有缺陷代码的"自然性"。收录于 第*38*届国际软件工程会议论文集。428–439。 doi:10.1145/2884781.2884848[35] Shuo Ren、Daya Guo、Shuai Lu、Long Zhou、Shujie Liu、Duyu Tang、Neel Sundaresan、Ming Zhou、Ambrosio Blanco 和 Shuai Ma。2020。CodeBLEU：代码合成的自动评估方法。doi:10.48550/arXiv.2009.10297[36] Daniel Russo、Sebastian Baltes、Niels van Berkel、Paris Avgeriou、Fabio Calefato、Beatriz Cabrero-Daniel、Gemma Catolino、Jürgen Cito、Neil Ernst、Thomas Fritz、Hideaki Hata、Reid Holmes、Maliheh Izadi、Foutse Khomh、Mikkel Baun Kjærgaard、Grischa Liebel、Alberto Lluch Lafuente、Stefano Lambiase、Walid Maalej、Gail Murphy、Nils Brede Moe、Gabrielle O'Brien、Elda Paja、Mauro Pezzè、John Stouby Persson、Rafael Prikladnicki、Paul Ralph、Martin Robillard、Thiago Rocha Silva、Klaas Jan Stol、Margaret Anne Storey、Viktoria Stray、Paolo Tell、Christoph Treude 和 Bogdan Vasilescu。2024。软件工程中的生成式人工智能必须以人为中心：《哥本哈根宣言》。《系统与软件杂志》216，112115（2024）。 doi:10.1016/j.jss.2024.112115[37] Caitlin Sadowski、Emma Söderberg、Luke Church、Michal Sipko 和 Alberto Bacchelli。2018。现代代码评审：Google 案例研究。收录于 第*40*届国际软件工程会议：软件工程实践论文集。181–190。doi:10.1145/3183519.3183525 [38] Markus Schnappinger、Arnoud Fietzke 和 Alexander Pretschner。2020。定义软件可维护性数据集：收集、汇总和分析软件可维护性的专家评估。收录于 第*36*届国际软件维护与演化会议论文集。278–289。[39] Arindam Sharma 和 Cristina David。2025。通过不确定性估计评估基于大语言模型的代码生成的正确性。doi: 10 48550 2502 11620 。/arXiv。。

[40] Dag Sjoberg 和 Gunnar Rye Bergersen。2023。。软件工程中的构念效度。《*IEEE* 软件工程汇刊》49，3（2023），1374–1396。doi:10.1109/TSE.2022.3176725[41] Dag Sjoberg、Aiko Yamashita、Bente Anda、Audris Mockus 和 Tore Dybå。2013。量化代码异味对维护工作量的影响。《*IEEE* 软件工程汇刊》39，8（2013），1144–1156。 doi:10.1109/TSE.2012.89

[42] Stripe。2018。《开发者系数：软件工程效率及其对全球*GDP*的*3*万亿美元影响》。技术报告。https://stripe.com/reports/developer-coefficient-2018。[43] Thoughtworks。。《技术雷达：当今技术格局的观点指南》。技术报告2025。https://www.thoughtworks.com/content/dam/thoughtworks/documents/radar/2025/04/tr_technology_radar_vol_32_en.pdf[44] Adam Tornhill 和 Markus Borg。2022。代码警报：代码质量的商业影响——对39个专有生产代码库的定量研究。收录于 第五届国际技术债务会议论文集。11–20。[45] Adam Tornhill、Markus Borg、Nadim Hagatulah 和 Emma Söderberg。2025。ACE：使用经验证的大语言模型重构实现自动化技术债务修复。收录于 第*33*届*ACM*国际软件工程基础会议伴会论文集。1318–1324。[46] Philip Walsh、Haritha Khandabattu、Matt Brasier 和 Keith Holloway。2025 年。人工智能代码助手魔力象限。技术报告 G00824517。Gartner。https://www.gartner.com/doc/reprints?id=1-2LVTG7RP&ct=250915&st=sb[47] Yanlin Wang、Wanjun Zhong、Yanxian Huang、Ensheng Shi、Min Yang、Jiachi Chen、Hui Li、Yuchi Ma、Qianxiang Wang 和 Zibin Zheng。2025 年。软件工程中的智能体：调查、现状与愿景。自动化软件工程 32，2（2025 年 8 月），70。doi:10.1007/s10515-025-00544-2[48] Miku Watanabe、Hao Li、Yutaro Kashiwa、Brittany Reid、Hajimu Iida 和 Ahmed E. Hassan。2025 年。关于智能体编码的使用：对 GitHub 拉取请求的实证研究。doi:10.48550/arXiv.2509.14745。/arXiv。。[49] Yisen Xu、Feng Lin、Jinqiu Yang、Tse-Hsun Chen 和 Nikolaos Tsantalis。2025 年。MANTRA：利用上下文检索增强生成和多智能体大语言模型协作增强自动化方法级重构。doi:10.48550/arXiv.2503.14340

## A 附录 A– 道德声明

尽管 CodeContests 数据集以宽松的 Apache 2.0 许可证发布，但其再利用引发了一些伦理讨论。DeepMind 将竞赛编程提交汇总到该数据集中，用于训练 AlphaCode，但目前仍不清楚参赛者在多大程度上明确同意将其用于这一二次用途。在我们的工作中，该数据集仅用于在公开研究背景下评估大语言模型输出，而不是训练新模型，因此减轻了相关的伦理担忧。我们注意到，这属于关于大语言模型训练伦理的更广泛讨论的一部分。

## B 附录 B – Claude 输出

以下是 Claude 对一批样本进行重构后生成的两份输出摘要示例。第一份描述了保守重构，第二份则描述了更大幅度的改进操作。

**Claude v2.0.13 — 保守重构后** **g**

应用于所有文件的重构原则：1. 描述性命名：将通用变量名（a、b、c、arr、l、n、m）替换为有意义的名称（array、count、total、num_tests 等）2. 整理导入语句：按逻辑对导入内容分组（先是标准库，然后是第三方库，最后是本地模块）3. 常量置顶：将所有常量（MOD、BUFSIZE、INF、EPSILON 等）移至文件顶部的导入语句之后 4. 适当间距：在函数、类和逻辑区块之间添加适当的空行 5. 保留功能：原有逻辑、算法和行为均保持不变

代码质量改进：
–提高了可读性和可维护性– 所有文件的代码结构保持一致 – 遵循整洁代码和 PEP 8 原则–功能没有任何变更 – 所有算法的运行方式完全一致

此次重构成功将 400 个独立的 Python 文件转换为更易于维护和阅读的代码，同时保留了 CLAUDE.md 中所规定的所有原有功能。

**Claude v2.0.13——经过更大胆的重构**

重构后的文件现在遵循整洁代码原则，可维护性得到提升，命名规范更加完善，相关功能也经过了恰当的类提取，整体代码组织更加一致。每个文件都保留了原有功能，同时可读性和可维护性显著提高。

应用于所有文件的关键重构改进：1. 改进命名–晦涩的变量名 –> 描述性名称– 单字母函数 –有意义的函数名– 更好的常量命名（大写_）2. 代码结构–将复杂逻辑提取到独立函数中–更好地实现关注点分离– 改进类组织 3. 符合 PEP8 规范– 统一命名规范（蛇形_命名）–适当的间距和缩进– 修复行长问题 4. 清除代码异味–魔法数字 –> 命名常量–简化复杂的条件语句–消除重复代码– 移除不必要的全局变量– 扁平化深层嵌套结构 5. 可维护性– 单一职责原则– 提高可读性– 更好地进行函数分解– 自我–说明的代码