arXiv:2603.17150v1 [cs.SE] 17 Mar 2026

意图形式化：人工智能智能体时代可靠编码面临的重大挑战

SHUVENDU K. LAHIRI，微软研究院，美国

智能体人工智能系统如今能够以惊人的流畅度生成代码，但一个根本性的问题仍然存在：生成的代码真的实现了用户的意图吗？非正式自然语言需求与精确程序行为之间的鸿沟——即意图鸿沟——一直困扰着软件工程，而人工智能生成的代码将这一问题放大到了前所未有的规模。本文认为，**意图形式化**——将非正式用户意图转换为一组可检查的形式化规范——是决定人工智能究竟会让软件变得更加可靠，还是仅仅让软件数量更加庞大的关键挑战。意图形式化提供了一个适用于不同场景可靠性需求的权衡谱系：从用于消除潜在误解的轻量级测试，到用于形式化验证的完整功能规格，再到能够自动合成正确代码的领域专用语言。核心瓶颈在于验证规约：由于除了用户之外不存在能够判定规约正确性的预言机，我们需要半自动化度量指标，通过轻量级用户交互和测试等代理制品，在有代码或无代码的情况下评估规格质量。我们综述了早期研究，这些研究展示了该方法的**潜力**：能够提升程序正确性的交互式测试驱动意图形式化，能够捕获以往方法遗漏的现实世界中的缺陷的人工智能生成的后置条件，以及能够根据非形式化规约生成可证明正确代码的端到端验证流水线。我们概述了尚待解决的研究挑战——超越基准测试实现扩展性、针对变更实现组合性、用于验证规约的度量指标、处理丰富逻辑，以及设计人机规约交互——这些挑战共同构成了一个横跨人工智能、编程语言、形式化方法和人机交互的研究议程。

关键见解

- 人工智能生成的代码在构造上具有合理性，但并非在构造上就具备正确性。意图鸿沟——用户的意图与程序的实际行为之间的距离——是可靠性的核心瓶颈。
- **意图形式化**——将非正式用户意图自动转换为可检查的规约——可以弥合这一鸿沟。它提供了适用于不同可靠性需求的权衡谱系：从轻量级测试，到用于形式化验证的完整功能规格，再到可据此自动合成正确代码的领域专用语言。
- 核心瓶颈在于**验证规约**：由于除了用户之外没有可用于判断规约正确性的判定预言机，我们需要半自动化度量指标，通过轻量级用户交互以及测试等代理制品，在有代码或无代码的情况下评估规约质量。
- 早期研究展示了其**潜力**——提升开发者正确性、发现新的漏洞，以及构建从非正式文本到正确代码的经验证管线——但关键**挑战**仍然存在：如何扩展到基准问题之外、在没有判定预言机的情况下验证规约，以及将意图形式化整合进智能体开发者工作流。
- 要推动意图形式化在生产系统中的规模化应用，需要持续投入人工智能、编程语言、形式化方法和人机交互领域的研究。

作者联系信息：S huvendu K. Lahiri，微软研究院，雷德蒙德，华盛顿州 ，美国，shuvendu@microsoft.com。

1 引言

氛围编程——Andrej Karpathy 创造的术语 [12]——概括了人工智能驱动的软件开发所呈现的新现实：开发者用自然语言描述需求，几乎不做或完全不做审查便接受人工智能生成的代码，并且“忘了代码的存在”。如今，智能体编程工具生态日益壮大，GitHub Copilot 编程智能体、Claude Code 等工具能根据简短提示词合成完整的函数、模块和系统，规划、编写代码、运行测试并不断迭代，而且往往能够自主完成。这正是意图鸿沟最纯粹的体现：用户有意图，却从不检查实现，完全依赖人工智能来弥合两者之间的鸿沟。

但一个根本问题仍未得到解答：生成的代码是否真的实现了用户的意图？

考虑一个简单的请求：“给定一个整数列表，删除重复项。”这是否意味着每个元素只保留一个副本（例如，`[1,2,3,2,4]` → `[1,2,3,4]`）？还是意味着删除所有出现不止一次的元素，只保留唯一元素（例如，`[1,2,3,2,4]` → `[1,3,4]`）？人类开发者会通过领域知识和交流来消除这类歧义。大语言模型则会根据训练数据进行统计模式匹配来消除歧义，却无法以用户的具体意图为依据。结果就是，代码看起来正确，但可能在不知不觉中偏离用户的真实需求。

例如，大语言模型对这个提示词的典型响应会生成：

Listing 1。若用户意图是删除所有存在重复项的数字，则这种做法看似合理，实则错误。

```
定义 移除_（数字）： 返回列表(字典.fromkeys（数字）
```

这样会按顺序保留每个元素的一份副本：remove_重复项(`[1,2,3,2,4]`) 返回`[1,2,3,4]`。但如果用户的意思是“移除所有出现不止一次的数字”，他们 [4] 期望`[1,3,4]`的结果应该是——2 应该被彻底移除。形式化的后置条件可以消除这种意图歧义：

代码清单 2. 描述“移除重复元素”意图的后置条件。

```
断言所有( 数字 .计数(x)  ==  1 对于 x 在 结果 )断言所有(x 在 结果 对于 x 在 数字 如果 数字 .
计数(x)  ==  1)
```

这个简单示例中的歧义揭示了一种更普遍的现象。意图鸿沟——用户的本意与程序实际行为之间的语义距离——在软件工程中一直存在（图 1），但人工智能从两个方面加剧了这一问题：

(1) 规模扩张，审查不足。人工智能生成代码的速度快于人类审查代码的速度。产出的代码与经过仔细检查的代码之间的比例正在迅速扩大，而传统的保障措施——代码审查、手动测试——已难以跟上。**(2) 看似可信，却不一定正确**。大语言模型生成的代码天生就显得可信——看起来没问题、能够编译，而且往往能通过几项测试——但它并非天生正确。细微错误藏在表面流畅的代码之下，因此人工智能生成的漏洞比手写代码中的漏洞更难发现。

我们认为，通往可靠的人工智能生成代码之路，并非改进代码生成，而是**意图形式化**：将非正式的用户意图自动转化为可检查的形式化规约。与其问“人工智能能写代码吗？”，我们不如问“人工智能能否帮助我们明确代码应该做什么，并验证代码是否确实做到了？”

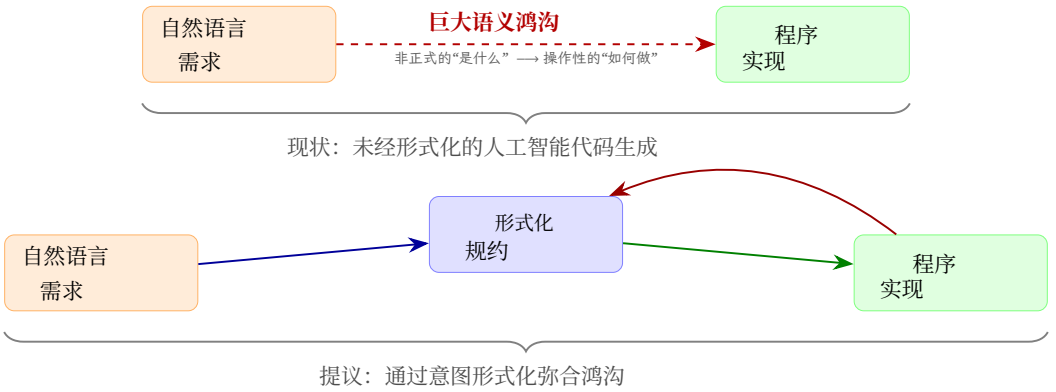


图 1. 软件开发中的意图鸿沟。**上图：** 传统的人工智能代码生成会将非正式且含糊的自然语言直接转换为可执行代码，留下巨大的语义鸿沟。**下图：** 形式化规约充当中间的“做什么”层，可缩小这一鸿沟，并通过测试和验证实现约束。

> 意图形式化提供了一个适用于不同可靠性需求的权衡谱系。一端是包含测试在内的轻量级规范，它们针对可能存在歧义之处——也就是不同大语言模型或智能体面对同一提示词时会生成语义不同代码的地方——并充当经济高效的**护栏**。中间是使用 Dafny、F* 和 Verus 等验证感知语言编写的完整功能规格，它们支持通过机器检查证明正确性。另一端则是领域专用语言（DSL），它们本身就是完整的规约，可据此自动生成可证明正确的代码。

本文提出了一个意图形式化框架，综述了展示其潜力的早期研究，并概述了界定未来十年研究议程的未决问题。本文旨在阐明研究界和产业界持续投入意图形式化的必要性，并将其列为**优先级最高的研究重点**。

1.1 为什么是现在?

随着第一代神经代码生成模型（TiCoder [14]）问世，人们认识到意图形式化的必要性。OpenAI Codex（早在2021年）证明，大语言模型能够根据自然语言提示生成语法正确的代码，但也揭示了看似合理的代码很容易在不知不觉中偏离用户意图。在那个阶段，人工智能编程工具主要还是自动补全助手：它们提供单行代码或小段代码建议，而开发者仍掌握全部控制权，会在采纳每条建议前进行审查。人类仍是主要的代码作者；人工智能只是提升效率的辅助工具。

此后，整个领域发生了翻天覆地的变化。智能体编程工具能够自主编写、测试和调试完整功能。氛围编程 [12] 鼓励开发者描述意图，然后“让人工智能来处理”。人类的角色已从作者 转变为监督者，而在许多氛围编程场景中，更是成了人工智能生成的代码的被动使用者 。

这一转变带来了三项后果，使意图形式化变得至关重要：

(1) 人工审查正被绕过。当开发者几乎不审查或完全不审查就接受人工智能生成的代码时，传统上由人工提供的防护措施便会消失，无法再防止意图违背。形式化规范成为唯一 可扩展的机制，用于检查生成的代码是否符合用户意图。

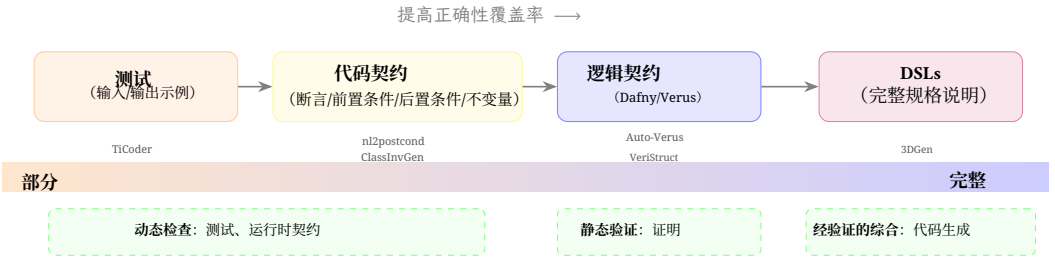


图 2. 意图形式化的谱系。**规约**（上排）从部分规约（测试）延伸至完整规约（领域专用语言）——它们都是形式化且可检查的，只是在正确性覆盖率上有所不同。测试和代码契约通过动态方式检查；逻辑契约需要通过程序验证器进行静态验证；领域专用语言则支持经验证的综合，生成构造即正确的代码。大语言模型可以帮助生成各个层级的产物。

(2) **攻击面已急剧扩大**。人工智能生成的代码正以前所未有的规模进入生产环境，包括安全关键型系统和安全敏感系统。解析器、身份验证模块或金融交易处理程序中的一个规约缺口，都可能造成严重后果。(3) **技术已趋成熟**。为氛围编程提供支持的同一批大语言模型，也能生成规约。验证基础设施（SMT 求解器、证明助手、类型系统）经过数十年的发展，已日臻成熟。现在，我们首次有望闭合整个流程：生成代码、生成规约，并验证二者是否相符——这一切都能在人工智能辅助工作流中完成。

2 什么是意图形式化?

我们将**意图形式化**定义为自动将非正式用户意图转换为一组形式化、可检查的程序规约这一问题。由此得到的规约构成一个表达能力逐渐增强的谱系（图 2）：

- **测试**（输入/输出示例）：具体的行为预期，例如，移除_重复项（`[1,2,3,2,4]`）应返回 `[1,3,4]`。
- **代码契约**（断言、前置条件/后置条件、不变量）：在运行时动态检查的可执行规约，从内联断言（例如 `assert all(nums.count(x)==1 for x in res)`）到函数级后置条件和类不变量。
- **逻辑契约**：在 Dafny [16], F* [23]、Verus [15] 等验证感知语言中编写的规约，使用量词、幽灵变量和递归谓词。这些规约需要程序验证器（例如 SMT 求解器）对所有可能的输入进行静态检查。
- **领域专用语言（DSL）**：采用专门表示法编写的完整形式化规范，可通过经验证的编译或综合自动生成有可证明正确性的代码。

关键在于，意图形式化并不 局限于验证感知语言——测试和代码契约适用于任何主流语言（Python、Java、C语言++、Rust）。Dafny 和 Verus 等验证感知语言可通过机器检查证明提供更强的保证，而领域专用语言则更进一步——作为完整规约，可据此自动综合出正确代码。不过，谱系中的前两个层级与语言无关，可以立即部署。

测试和代码契约可以通过运行程序进行动态检查。逻辑契约则需要通过程序验证器进行静态验证，这能提供更强的保证，但也要求更复杂的规约和工具链。领域专用语言处于谱系的最远端：其规约

足够完整，从而可以通过经验证的编译（第 3.4 节）自动合成 正确代码。一项关键洞见是，这些层级并非相互替代，而是相辅相成；任何一个层级取得进展，都能推动其他层级进步。例如，测试可以验证后置条件，后置条件可以指导不变量发现，而不变量则可以为完整证明奠定基础。[1]

> **两个重要区别**。并非自动形式化。意图形式化不同于自动形式化——后者是将完整的自然语言规范转换为形式逻辑 [25]。自动形式化力求忠实地转换完整的源文本；意图形式化则提供一个成本效益各异的谱系：从消除本质上不完整的自然语言提示中最容易产生歧义的属性，到采用领域专用语言自动合成正确代码。
>
> 与规范驱动开发互为补充。GitHub Spec Kit [8]等工具以自然语言需求为核心组织人工智能编码，从而提升可追溯性。但这些规约仍是非正式的，无法检查。意图形式化通过生成形式化且可检查的规约，并可针对生成的代码进行机械验证，弥补了这一缺口。

2.1 规约与验证

区分规约和验证十分重要。规约描述代码应该做什么；验证则检查代码是否确实做到了这一点。

- **测试**通过有限数量的输入来验证规约。测试轻量且适用范围广，但本质上并不完整：通过所有测试，并不能保证代码对未见过的输入也正确。
- **运行时检查**会在生产环境中验证每次执行时的规约。后置条件和断言会在运行时进行评估，在违规发生时及时捕获。这比离线测试提供更强的覆盖率，但会产生运行时开销，并且只有在错误被触发时才能检测到。
- **证明**会针对所有可能的输入验证规约。它们由 SMT 求解器 [2]或证明助手生成，能够提供数学保证，但需要更丰富的规约和更复杂的自动化。

意图形式化关注的是规约侧：一旦规约准备就绪，现有的验证基础设施——测试运行器、SMT 求解器、证明助手——就可以对其进行检查。

> 当前的一个关键瓶颈，是缺乏可供验证的形式化规范。尽管验证技术本身仍在不断发展，规范却是前提条件——没有规范，验证工具就只能闲置。意图形式化提供了一个权衡谱系：即使只针对可能产生歧义的环节制定目标明确的规范，也能充当经济高效的护栏 [14], ；随着可靠性需求的提升，团队还可以进一步投入，通过完整功能规格开展形式化验证。

3 意图形式化的早期研究

意图形式化并非一种空想的研究方向。多条相互独立的早期研究路线——主要围绕基准问题展开——为这一方法的潜力提供了确凿证据。我们从四个维度梳理这些证据，依次从个体能力探讨到端到端系统。

[1]意图形式化并不局限于代码生成。即使是测试预言机——即单个测试输入的预期输出——也是意图形式化的一种形式 [3]。

3.1 大语言模型能够生成有意义的规约

通过自然语言描述提示的大语言模型能够生成后置条件——可执行的断言，用于约束任意输入下的函数输出 [4]。在 Defects4J 基准测试 [10]（涵盖大型 Java 项目中的数百个真实漏洞）上，大语言模型生成的后置条件发现了八分之一的真实漏洞，其中包括经典 Daikon 不变式检测器遗漏的漏洞 [5]。GPT-4 生成的后置条件在可靠性和完备性方面的得分显著高于 GPT-3.5 或 CodeLlama，这表明规格质量会随着模型能力的提升而提高。

除了函数级后置条件之外，ClassInvGen [20] 还为 C语言++ 数据结构合成类不变量——这些关键属性将整个模块紧密联系在一起。一个精心选择的类不变量就能取代数百条函数级规约，使审查变得切实可行，并揭示实践者最为关注的关键非平凡属性。ClassInvGen 在这项任务上的表现优于直接使用大语言模型提示词以及 Daikon。VeriStruct [22] 则进一步扩展到 Verus 中的整个数据结构模块，验证了 11 个模块中几乎所有函数，这些模块包括链表、哈希映射和 B 树。

这些结果表明，大语言模型能够生成体现真正语义理解、而不只是句法模式的规约。

但一个更根本的问题随之而来：我们如何衡量生成的规约是否足够好？

3.2 衡量规约质量

意图形式化面临的一项根本挑战是除了用户之外，没有其他判定预言机可以验证规约的正确性。代码可以依据规约进行测试，但谁来测试规约？与代码不同，测试套件可以提供独立检查，而生成的规约没有可供比对的真实标准——用户的意图只存在于用户脑海中。因此，**验证规约**是最亟待解决的开放问题之一。

我们主张采用基于两项属性的自动化指标 [4, 13]：

- **可靠性**：规约与正确行为一致——不会拒绝有效的实现。
- **完备性**：规约具有区分能力——会拒绝不正确的实现。

一种方法 [13] 仅使用测试（输入/输出对）来将这些属性转化为可操作指标，而无需代码本身：

- 如果 S在每项测试（i, o） $\in T$中均得到满足，则规约 S相对于测试套件 T是可靠的。
- 如果对于每个测试（i， o） $\in T$，当输出 o 被替换为变异后的输出 o_i时， S测试失败，那么该规约 S就是完备的。为兼顾非确定性（即对于给定输入，可能有多个有效输出），完备性以规约能够检测到的输出变异所占比例来衡量。

可靠性的主要挑战在于构建一个穷尽的测试集合：一项规约即使通过了少数几项测试，仍可能会拒绝未测试输入上的有效行为。完备性则更难：它不仅要求规约准确捕捉用户的意思，还要求输出变异体现恰好用户的真实意图，以及自然的错误——也就是人类（如今越来越多地包括人工智能）实际会犯的那类错误。

由于这两种属性都定义为依据测试来评估规约，因此可以在编写任何实现之前进行检查 [4, 13]。对于简单的可执行断言，这种评估很直接；但验证感知语言中的规约往往包含量词、递归谓词和幽灵变量，无法直接执行。Lahiri 等人 [13] 提出了新的符号技术，通过具体测试输入来评估这类丰富的规约，从而即使面对

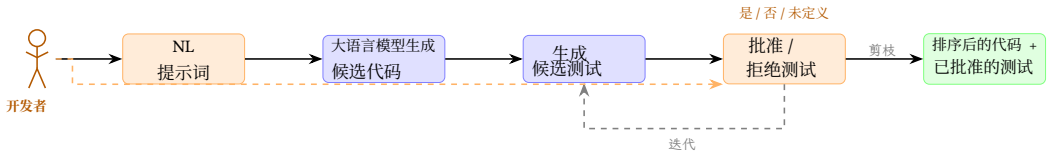


图 3. TiCoder 的测试驱动用户意图形式化交互工作流。开发者提供自然语言提示；大语言模型生成候选代码和测试。用户通过反复批准或拒绝测试来筛选并排列候选项。这使人工智能代码的正确评估率从 40% 提高到 84% [7]。

复杂的逻辑契约，也能进行自动化可靠性与完备性检查。这为规格质量提供了客观、可复现的衡量方法——正如自然语言到代码研究借助 HumanEval [17]等基于测试的基准测试取得进展一样。基于测试的评估是一种有前景的方法；其他技术——符号分析、基于属性的测试和有针对性的用户交互——也可能进一步加强规格验证。

以自动化度量指标的强大作用为例：GPT-4 生成的 Dafny“公共元素”规约被专家评审员标记为“强”[18]：

代码清单 3. 一份被专家评审员标记为“强”的 Dafny 规约——但自动化完备性指标揭示它并不完整 [13]。

```
确保 forall x::x属于 结果 ==>(InArray(a,x)&&InArray(b,x))确保 forall i,j:: 0<=i<j
<|结果 | ==>结果[i]!=结果[j]
```

这表示结果中的每个元素都出现在两个输入中，且结果中没有重复项——听起来不错，但并不完整。自动符号测试 [13] 发现，蕴含关系（==>）应当是双条件（<==>）：否则，空列表就能轻易满足该规约。修正后的版本：

清单 4. 使用双条件的修正规约。

```
确保 对所有 x::x in 结果 <==> (InArray(a, x) && InArray(b, x))
```

在整个评估过程中，自动化指标发现了三个误标和两个因复制粘贴错误而产生的不一致规约，而人工标注 [13]未能发现这些问题。无法衡量的事物，就无法改进，自动化指标对于大规模推进意图形式化至关重要。

这些度量指标也有助于下游的证明自动化。Auto-Verus 系统 [1] 使用可靠性和完备性度量指标 [13] 筛选大语言模型为 Rust/Verus 程序生成的规约和证明，通过自进化循环构建高质量训练数据，并实现了3.6× 高于 GPT-4o 零样本表现的证明准确率。

> 更丰富、质量更高的规约能够增强人工智能生成证明的能力。

有了可靠的度量指标，我们就可以探究规约是否能提升代码生成本身，而不仅仅是在事后对其进行评估。当必须咨询用户时，应优化交互方式，以便通过每次用户查询获得尽可能多的正确规约——TiCoder 正是这一原则的体现。

3.3 交互式意图形式化

TiCoder 系统 [7, 14] 在代码生成过程中以交互方式使用意图形式化（图 3）。

TiCoder 不会直接接受大语言模型生成的任何代码，而是生成候选测试，优先关注存在歧义的地方——也就是不同代码候选方案会产生不同输出的输入——并请用户将每项标记为“是”“否”或“未定义”。已批准的测试会通过

代码执行来筛除错误候选项。这体现了按需付费理念：TiCoder 从谱系中最具成本效益的一端入手，优先针对最可能揭示大语言模型的理解与用户意图存在偏差的测试。

> **侧栏：TiCoder 交互**
> **提示词：**“找出两个列表中的共有元素。”
> **生成的测试：**
> (1) common([1,2,3],[2,3,4]) == [2,3] 用户：是
> (2) common([1,2,2],[2,2,3]) == [2,2] 用户：不
> 通过拒绝测试 2，用户明确表示结果应为一个集合，而非多重集。TiCoder 会剔除保留重复项的候选结果。

一项针对代码生成基准测试 [7] 的小规模对照研究纳入了 15 位专业开发人员，研究发现：使用 TiCoder 时，开发人员正确评估人工智能生成的代码的频率约为未使用时的两倍（$p < 0.001$）；认知负荷显著降低（$p = 0.007$）；已批准的测试会保留为回归测试；并且大多数参与者更喜欢 TiCoder。在这些基准测试中，只需适度进行意图形式化——批准少量测试——就能显著提升正确性。

3.4 端到端经验证的流水线

在谱系的另一端，3DGen 系统 [6] 展示了 DSL 层面的意图形式化，此时规约足够完整，可以自动生成代码。3DGen 使用多智能体人工智能架构，将非正式的 RFC 文本转换为 3D领域特定语言中的形式化规约，并通过符号测试生成提供反馈，以便迭代细化。这些经验证的 3D 规约经由 EverParse [24] 编译为可证明正确且内存安全的 C语言或 Rust 二进制解析器——规约就是程序，并由经验证的综合实现转换。该系统已为 20 种标准网络协议格式（DNS、TLS扩展、QUIC）生成经验证的解析器，展现了完整谱系的潜力：从非正式文本，经由 DSL，直至可证明正确、可部署的代码。

4 研究议程

前述早期研究在基准问题上展现了潜力，但也揭示出我们距离通用解决方案还有多远。七个尚未解决的问题构成了研究前沿，涵盖人工智能能力、形式化方法基础设施、人机交互设计和软件工程实践。

*从基准测试迈向真实世界系统。*当前成果针对的是自包含算法函数。真实世界的软件具有副作用、可变状态、并发和复杂的依赖关系。对于异步事件处理程序或机器学习流水线，“后置条件”意味着什么？我们需要针对真实世界意图形式化的基准测试、度量指标和规约惯用法。

变更意图与组合性。 在实践中，大多数软件开发都涉及修改现有代码，而非从头编写。这里的意图不仅包括自然语言变更描述——“修复这个错误”“添加缓存”“处理空输入边界情况”——还包括定义当前行为的现有源代码、测试和规约。因此，意图形式化必须明确行为应如何改变，并与现有规约相结合。与之密切相关的挑战是代码转换——例如，将遗留 C 代码库迁移到 Rust。

SpecTra [19] 等早期工作表明，从源代码生成中间非形式化规约，并以此指导转换，能显著提高正确性。对待转换代码的意图进行形式化，对于实现可靠的代码转换至关重要。

以经济高效的方式识别需要澄清的内容。对于从业者而言，规约的价值取决于它能预防多少漏洞。因此，我们需要能够按预期影响对规约进行排序的度量指标，尤其是在将规约呈现给用户进行验证时。TiCoder 通过生成多样化的代码候选方案，并针对候选方案存在分歧的输入生成测试，迈出了第一步；但这需要对合理实现的空间进行采样，而对于大型代码块或多函数任务，这一过程的成本会变得很高。我们需要可扩展的方法，在不穷举候选程序的情况下，依据规约预防漏洞的价值确定优先级。

用于规约验证的自动化指标。除了用户之外，没有其他判定预言机能够判断规约是否正确，因此验证生成的规约是一项首要挑战。要取得进展，就必须结合多种互补信号：将测试和变异分析作为自动化代理 [4,13], ，通过有针对性的用户反馈解决自动化无法判定的歧义，并利用代码、文档字符串和形式化注释等不同工件进行交叉核验 [21]。

丰富逻辑与量词。 验证感知语言使用量词、递归谓词和幽灵变量。大语言模型难以处理这些结构，而现有验证器在针对复杂具体测试输入展开递归谓词时存在根本性局限，因此难以自动评估可靠性与完备性 [13]。

用于规约的人机交互。 TiCoder 的批准/拒绝循环显著提升了基准问题上的正确性，但真实世界中的规约工作需要更丰富的交互方式：以自然语言解释形式化属性、提供经过置信度校准的建议，以及使用规约模板——这是一个鲜少探索的人机交互设计领域。

融入开发者工作流。 意图形式化必须自然地融入现代开发者工作流——从问题创建开始，在编写代码之前用规约记录预期行为；到代码审查阶段，让规约向审查者呈现意图；再到 CI/CD 流水线，通过智能体工作流 [9] 持续发现规约，并根据不断演进的需求对其进行验证。在氛围编程时代，人类可能从不检查代码，规约因此成为人类意图与机器行为之间的主要接口 ——这使得这种集成不仅值得追求，而且不可或缺。

5 相关工作

意图形式化借鉴并拓展了几个成熟领域。**规约挖掘**：Daikon [5] 等经典工具能够从执行轨迹中推断不变量，但无法捕捉超出已观察行为范围的用户意图；基于大语言模型的规约生成通过推理预期的语义，突破了这一局限。**基于大语言模型的代码生成**：HumanEval [17] 等基准测试通过 pass@k 评估代码正确性，却不评估规约质量；意图形式化将评估目标从“代码能否通过测试？”转变为“规约是否捕捉了意图？”。**形式化验证**：SMT 求解器 [2], 、证明助手以及验证感知语言（Dafny [16],、Verus [15]）为检查规约提供了成熟的基础设施，却无法生成规约——意图形式化补上了缺失的输入。**用于验证的大语言模型**：近期利用大语言模型进行程序验证 [11] 以及 Dafny 规约合成 [18] 的研究，主要致力于让给定程序成功通过验证，而非为尚不存在的代码捕捉用户意图。

6 结论

人工智能生成代码的时代已经到来；可靠的人工智能生成代码时代尚未到来。我们介绍了早期研究，表明**意图形式化**——通过形式化规范明确表达用户意图，使其可检查、可强制执行——是一个很有前景的方向。大语言模型能够生成涵盖完整形式化谱系的规约。自动化指标能够以达到或超过专家水平的表现评估规约质量。交互式形式化显著提升了开发者在基准问题上的正确率。端到端流程能够从非正式文本生成经过验证的代码。而规约质量则有助于实现证明自动化。

这些尚待解决的挑战——突破基准问题的规模限制、形式化变更意图、确定如何以经济高效的方式明确需要澄清的事项、用于规约验证的自动化指标、处理丰富逻辑、设计人机交互，以及实现变更的组合性——各自都是规模宏大的研究方向。我们呼吁学界将意图形式化视为**首要任务**：制定专门的基准测试，推动人工智能、编程语言、形式化方法和人机交互研究者开展跨学科合作，并持续投入资源。意图鸿沟是瓶颈；能否弥合这一鸿沟，将决定人工智能究竟会让软件更可靠，还是仅仅让软件数量更多。

致谢。本文汇集了许多合作者的贡献，包括 Nikolaj Bjorner、Sarah Fakhoury、Saikat Chakraborty、Markus Kuppe、Shan Lu、Tahina Ramananandro、Nikhil Swamy、Aaditya Naik、Georgios Sakkas、Madeline Endres、Elizabeth Dinella、Todd Mytkowicz、Madanlal Musuvathi，以及微软研究院 RiSE 团队的许多其他成员。本文所述内容取材于发表于 FSE 2024 [4],FMCAD 2024 [13], IEEE TSE 2024 [7],ICSE 2025 [6], ICLR 2025 [1], SAIV 2025 [20], 和 TACAS 2026 [22] 的研究成果。

参考文献

[1] Tianyu Chen、Shuai Lu、Shan Lu、Yeyun Gong、Chenyuan Yang、Xuheng Li、Md Rakib Hossain Misu、Hao Yu、Nan Duan、Peng Cheng、Fan Yang、Shuvendu K. Lahiri、Tao Xie 和 Lidong Zhou。2025。通过自我演进为 Rust 代码自动生成证明。载于国际学习表征会议（*ICLR*）论文集。[2] Leonardo de Moura 和 Nikolaj Bjørner。2008。Z3：一种高效的 SMT 求解器。系统构造与分析工具和算法国际会议（*TACAS*）（2008）。[3] Elizabeth Dinella、Gabriel Ryan、Todd Mytkowicz 和 Shuvendu K. Lahiri。2022。TOGA：一种用于生成测试预言机的神经网络方法。载于国际软件工程会议（*ICSE*）论文集。[4] Madeline Endres、Sarah Fakhoury、Saikat Chakraborty 和 Shuvendu K. Lahiri。2024。大语言模型能否将自然语言意图转换为形式化方法后置条件？载于*ACM* 软件工程基础国际会议（*FSE*）论文集。[5] Michael D. Ernst、Jake Cockrell、William G. Griswold 和 David Notkin。1999。动态发现可能的程序不变量，以支持程序演进。载于第 *21* 届国际软件工程会议（*ICSE*）论文集。[6] Sarah Fakhoury、Markus Kuppe、Shuvendu K. Lahiri、Tahina Ramananandro 和 Nikhil Swamy。2025。3DGen：人工智能辅助生成可证明正确的二进制格式解析器。载于第 *47* 届国际软件工程会议（*ICSE*）论文集。[7] Sarah Fakhoury、Aaditya Naik、Georgios Sakkas、Saikat Chakraborty 和 Shuvendu K. Lahiri。2024。基于大语言模型的测试驱动交互式代码生成：用户研究与实证评估。*IEEE* 软件工程汇刊50，9（2024），2254–2268。[8] GitHub。2025。从模糊到精准：面向 GitHub Copilot 的 Spec Kit。https://developer.microsoft.com/blog/spec-driven-development-spec-kit。访问时间：2026 年 3 月。[9] GitHub。2026。使用 GitHub 智能体工作流自动化仓库任务。https://github.blog/ai-and-ml/automate- repository-tasks-with-github-agentic-workflows/。访问时间：2026 年 3 月。[10] René Just、Darioush Jalali 和 Michael D. Ernst。2014。Defects4J：现有故障数据库，助力 Java 程序的受控测试研究。载于国际软件测试与分析研讨会（*ISSTA*）论文集。[11] Adharsh Kamath、Nausheen Mohammed、Aditya Senthilnathan、Saikat Chakraborty、Pantazis Deligiannis、Shuvendu K. Lahiri、Akash Lal、Aseem Rastogi、Subhajit Roy 和 Rahul Sharma。2024。利用大语言模型进行程序验证。载于计算机辅助设计形式化方法国际会议（*FMCAD*）论文集。

[12] Andrej Karpathy。2025。氛围编程。*X*（原 *Twitter*）（2025）。https://x.com/karpathy/status/1886192184808149383。访问时间：2025 年 2 月。[13] Shuvendu K. Lahiri。2024。评估大语言模型驱动的用户意图形式化在验证感知语言中的应用。载于计算机辅助设计形式化方法国际会议（*FMCAD*）论文集。[14] Shuvendu K. Lahiri、Sarah Fakhoury、Aaditya Naik、Georgios Sakkas、Saikat Chakraborty、Madanlal Musuvathi、Piali Choudhury、Curtis von Veh、Jeevana Priya Inala、Chenglong Wang 和 Jianfeng Gao。2022。通过测试驱动的用户意图形式化实现交互式代码生成。*arXiv* 预印本 *arXiv:2208.05950*（2022）。[15] Andrea Lattuada、Travis Hance、Chanhee Cho、Matthias Brun、Isitha Subasinghe、Yi Zhou、Jon Howell、Bryan Parno 和 Chris Hawblitzel。2023。Verus：使用线性幽灵类型验证 Rust 程序。*ACM* 编程语言论文集（*OOPSLA*）（2023）。[16] K. Rustan M. Leino。2010。Dafny：一种用于函数正确性的自动程序验证器。程序设计、人工智能与推理逻辑（*LPAR*）（2010）。[17] Jiawei Liu、Chunqiu Steven Xia、Yuyao Wang 和 Lingming Zhang。2023。由 ChatGPT 生成的代码真的正确吗？对用于代码生成的大语言模型进行严格评估。载于神经信息处理系统进展（*NeurIPS*）。[18] Md Rakib Hossain Misu、Cristina V. Lopes、Iris Ma 和 James Noble。2024。迈向经验证的 Dafny 方法的人工智能辅助综合。载于*ACM* 软件工程基础国际会议（*FSE*）论文集。[19] Vikram Nitin、Rahul Krishna 和 Baishakhi Ray。2024。SpecTra：通过生成多模态规约增强语言模型的代码转换能力。*arXiv* 预印本 *arXiv:2405.18574*（2024）。[20] Chuyue Sun、Viraj Agashe、Saikat Chakraborty、Jubi Taneja、Clark Barrett、David Dill、Xiaokang Qiu 和 Shuvendu K. Lahiri。2025。ClassInvGen：使用大语言模型进行类不变量综合。载于人工智能验证国际研讨会（*SAIV*）论文集（计算机科学讲义，第 *15947* 卷）。Springer。[21] Chuyue Sun、Ying Sheng、Oded Padon 和 Clark Barrett。2024。Clover：闭环可验证代码生成。载于人工智能验证国际研讨会（*SAIV*）论文集（计算机科学讲义）。Springer。[22] Chuyue Sun、Yican Sun、Daneshvar Amrollahi、Ethan Zhang、Shuvendu K. Lahiri、Shan Lu、David Dill 和 Clark Barrett。2026。VeriStruct：Verus 中数据结构模块的人工智能辅助自动化验证。载于系统构造与分析工具和算法国际会议（*TACAS*）论文集。[23] Nikhil Swamy、Cătălin Hriţcu、Chantal Keller、Aseem Rastogi、Antoine Delignat-Lavaud、Simon Forest、Karthikeyan Bhargavan、Cédric Fournet、Pierre-Yves Strub、Markulf Kohlweiss、Jonathan Protzenko、Jean-Karim Zinzindohoué 和 Santiago Zanella-Beguelin。2016。F* 中的依赖类型与多单子效应。载于*ACM SIGPLAN* 编程语言原理研讨会（*POPL*）论文集。[24] Nikhil Swamy、Tahina Ramananandro、Aseem Rastogi、Irina Spiridonova、Haobin Ni、Dmitry Malloy、Juan Vazquez、Michael Tang、Omar Cardona 和 Arti Gupta。2022。利用形式化证明的二进制格式解析器强化攻击面。载于*ACM SIGPLAN* 编程语言设计与实现大会（*PLDI*）论文集。[25] Yuhuai Wu、Albert Q. Jiang、Wenda Li、Markus N. Rabe、Charles Staats、Mateja Jamnik 和 Christian Szegedy。2022。利用大语言模型进行自动形式化。载于神经信息处理系统进展（*NeurIPS*）。