函数式编程课程背景下的大语言模型评估：一项全面研究

Yihan Zhang[a] Brigitte Pientka[a] Xujie Si[b]a 加

拿大蒙特利尔麦吉尔大学 b 加拿大多伦多多伦多大学

摘要 大语言模型（LLM）正在改变学习者在课堂之外获取知识的方式。以 Java 或 Python 等高资源编程语言为例，先前研究表明，大语言模型似乎擅长为计算机科学导论课程生成简短而简单的问题。

本文评估大语言模型在低资源编程语言 OCaml 的教育场景中的有效性。具体而言，我们构建了三个基准测试集，全面评估 9 个最先进的大语言模型：1) λCodeGen（包含自然语言编程作业题的基准测试集）；2) λRepair（包含来自真实学生提交作业的语法、类型和逻辑错误程序的基准测试集）；3) λExplain（包含有关编程理论概念的自然语言问题的基准测试集）。我们使用 OCaml 编译器和自动评分器，根据正确性为各大语言模型的回答评分。此外，我们还采用人工评分来评估回答质量，使评估超越了常见的评估方法。

我们的研究表明，排名前三的大语言模型在典型函数式编程课程的所有任务上都表现有效，不过与它们在 Python 和 Java 入门编程问题上的成功相比，它们在低资源环境下解决的作业问题要少得多。大语言模型的优势在于纠正语法和类型错误，以及为基础概念性问题生成答案。尽管大语言模型在某些领域可能尚无法与专用语言工具相媲美，但它们作为多种编程语言的一站式工具所带来的便利，可能超过更专用系统的优势。

我们希望我们的基准测试能够发挥多重作用：评估大语言模型不断演进的能力，帮助授课教师提高学生对大语言模型所生成解决方案之局限性的认识，并让编程语言研究人员了解将领域专属推理融入大语言模型、以及为低资源语言开发更强大的代码合成与修复工具的机会。

ACM CCS 2012

- **软件及其工程** → **函数式语言**; 实证软件验证
- ；**计算方法** → **语言资源**；**通用与参考** → 计算标准、*RFC* 和指南；
-

关键词计算机科学教育、函数式编程、大语言模型

《编程的艺术、科学与工程》

提交日期2025年10月2日发表日期2026年2月15日DOI

10.22152/programming-journal.org/2026/11/5© Yihan Zhang a

品采用“知识共享署名 4.0 国际”许可协议授权，载于《编程的艺术、科学与篇文章；共31页。

1 引言

大语言模型（LLM）很容易被程序员和学生使用。学生使用大语言模型编写代码、修复代码 [30, 41, 57]，以及解释程序和课程概念 [20, 25, 28, 34, 42]。不管我们是否喜欢，大语言模型正在改变学生获取知识和寻求解释的方式。

然而，对学生而言，大语言模型是一把双刃剑。大语言模型似乎擅长生成简短、简单的答案，但也可能给出错误答案，而学生却以为这些答案是正确的。Kabir、Udo-Imeh、Kou 和 Zhang [24] 发现，ChatGPT 在 Stack Overflow 上针对编程问题生成的回答中，超过 50 % 是错误的，且超过 77% 的回答冗长啰唆、不够简洁。然而，在超过 35 % 的情况下，学生仍常常更青睐 ChatGPT回答，因为其内容全面、措辞流畅，他们因此忽略了其中的错误信息。过度依赖大语言模型可能给学生和授课教师带来挑战，尤其是在更高级的课程中。学生不仅会发现网上关于某些主题的资料较少，而且由于数据稀缺，大语言模型也可能无法得到有效训练。

本文旨在了解大语言模型在一门二年级函数式编程课程中的效果；该课程使用资源较少的编程语言 OCaml。课程涵盖从基础到高级的函数式编程技术，并向学生介绍程序语言设计中的理论概念。这与现有研究形成对比，后者大多聚焦于在计算机科学入门课程中使用主流编程语言 [12]。

我们重点关注以下五个研究问题：

RQ1 大语言模型能否正确且简洁地完成函数式编程作业？**RQ2** 大语言模型能否正确且简洁地修复函数式程序中的语法、类型和逻辑错误？**RQ3** 大语言模型能否正确且简洁地解释与程序语言设计相关的理论概念？**RQ4** 大语言模型生成正确且简洁答案的能力是否会因不同问题类型而异——从生成和修复代码到解释理论概念？

RQ5 问题的难度等级如何影响大语言模型的回答？

为回答这些问题，我们构建了三个不同的基准测试集：1） λCodeGen（一个包含自然语言编程作业题的基准测试集），2）λRepair（一个包含取自实际学生提交作业、带有语法、类型和逻辑错误的程序的基准测试集），以及3） λ解释（一个包含有关编程理论概念的自然语言问题的基准测试集）。与HumanEval[9] 和 MBPP [3]等常用数据集相比，我们的基准测试集要小得多，因为我们还对大语言模型生成的回答进行了定量和人工评估。尽管规模较小，它仍然具有代表性，因为它涵盖了函数式编程的关键范式。

第一个基准测试集 λCodeGen 包含 10 个多任务编程作业，每个作业由若干以自然语言描述和类型规范给出的编程任务组成。涵盖的内容范围从基本概念，例如

模式匹配、数据类型、map 等基础高阶函数，到延续等更高级的编程技术。它们还涉及编程语言中的高级概念，例如依赖于玩具语言中约束变量、替换、静态语义和操作语义的形式化定义的程序。因此，我们的基准测试集针对的是特定的函数式编程任务，而非 Python、Java 或 C 等资源丰富的语言中的通用程序。

第二个基准测试集 λRepair包含150个 OCaml 程序，这些程序取自麦吉尔大学二年级函数式编程课程中学生的真实错误。它涵盖三种不同类型的错误：语法错误（λRepairSyntax（50））、类型错误（λRepairType（50））和逻辑错误（λRepairProg（50））。这与 QuickBugs [56] 等使用合成数据的基准测试集形成对比。

我们的最后一个基准测试集 λExplain 包含50道题目，取自真实课前练习题和考试题语料库，聚焦于（函数式）编程概念。这些题目关注的概念包括覆写、遮蔽、值调用与名调用求值、状态存在时的求值等。

我们使用这些基准测试来评估大语言模型在不同任务上的表现——从生成代码（**研究问题1**）、修复有缺陷的代码（**研究问题2**），到提供解释（**研究问题3**）。此外，我们还比较大语言模型在不同类型和难度等级的问题上的能力（**研究问题4**和 **研究问题5**）。

为评估所生成与所修复代码的正确性，我们使用 OCaml 编译器与自动评分器。此外，我们开展人工评估，以评价模型输出的不仅是正确性，还有 质量 。

在本研究中，我们考察了 9 个最先进的大语言模型，其中包括 GPT-4o 等广泛使用的模型。近期调查与研究表明，ChatGPT/GPT-4o 仍然是学生最常用的 AI 工具[1] ([17])。我们观察到，GPT-4o 始终处于顶尖梯队。

在 λCodeGen 上，GPT-4o 和 o3-mini 等顶尖模型有 74% 的时间以精通级别生成代码，远高于早期研究中的结果。广泛使用的 GPT-4o 模型位列前三，以精通级别解决了 69.8% 的问题。这些数字明显低于入门级高资源 Python/Java 基准测试上的表现 [3, 9], ，原因有二：（1）我们的求值不仅考虑正确性，还考虑输出质量，而大多数先前研究仅关注正确性；（2）与标准的函数级任务不同，我们的基准测试包含低资源语言中的多任务 问题。为与领域专用工具提供基线比较，我们还在 λCodeGen 基准测试集上对 BURST [35]，（一个面向 OCaml 的代码合成系统）进行求值。BURST 仅为 11.3% 的问题生成正确的解决方案，表现显著差于通用大语言模型。

请注意，我们提及高资源语言的基准测试集，是为了提供参考，而非直接比较。基准结构和模型训练方面的差异，使得严格的跨语言结论无法成立。因此，我们引用这些基准测试集

[1] https://www.digitaleducationcouncil.com/post/digital-education-council-global-ai-student-survey-2024。访问于2025-02-10。

来说明一种普遍趋势：在热门高资源语言上表现出色的大语言模型，在 OCaml [6, 61] 这类低资源语言上的表现可能较弱。这样的表述有助于读者结合背景理解我们的研究结果，同时不会暗示我们进行了正式的跨语言比较。

我们的结果还揭示了局限性：在受测的大语言模型中，40 % 的模型在不到 41 % 的问题上达到精通级别，平均表现低于 50 %。与基础编程任务相比，所有模型在需要理论实现的抽象概念方面都更为吃力。从自然语言生成代码比修复代码更具挑战性：大语言模型在语法/类型错误上的表现比在逻辑错误或完整代码生成方面高出 5 %。此外，我们观察到，小型模型在代码修复问题上的表现有所提升（Llama3.1 8B 和 Qwen2.5 7B 的精通级别解决方案约增加 20 %）。总体而言，各模型之间的表现差距在 λ解释 上进一步扩大：GPT-4o 能解决 58.4 % 的问题，而 Llama3.1 8B 只能解决 16.8 %。综上，这进一步表明，大语言模型的训练数据至关重要，而通用大语言模型在处理更高级、更复杂的低资源编程概念时表现不佳。

这些结果对不同群体都有启示。对于学生而言，结果凸显了培养批判性评估大语言模型输出的能力的重要性，而不是不加批判地依赖它们——这既是伦理问题，也是实际问题。对于授课教师而言，我们的基准测试集提供了一种具体方式，既能帮助人们认识到大语言模型的优势与不足，也能设计超越传统问题求解的评估，转而侧重批判性评估、调试和验证。例如，他们可以要求学生设计一些问题，诱使模型给出错误答案 [15]。对于编程语言（PL）社区而言，我们的发现既凸显了大语言模型的适应能力，也揭示了当前领域专用工具（如 BURST）的局限。这表明，可以将规范感知推理融入大语言模型，结合其灵活性与综合技术。更广泛地说，我们的基准测试集推动人们开发更强大的综合与修复工具，以应对低资源语言以及更大、更复杂的程序，而编程语言研究人员可以在其中作出独特贡献。

2 方法

本节介绍麦吉尔大学一门二年级函数式编程课程中的编程问题和概念性问题。选修这门课程的学生大多处于大学二年级，甚至三年级或最后一年。我们还介绍了对大语言模型的求值方式和评分政策。

2.1 Benchmark λCodeGen：作业编程问题（10 项作业）

本次评估采用麦吉尔大学2022年秋季二年级函数式编程课程中的编程作业题。这门课程共有10份作业，包含53项任务。表1简要概述了每份作业的内容及词元数量。每道题的平均词元数为

■ **表 1** 2022年秋季的编程作业题

HW	# 词元	# 任务	主题	难度
1	558	3	基础、递归与模式匹配	B
2	993	8	基础递归数据类型与尾递归函数 更复杂的递归数据类型与递归	B AP
3	783	7	高阶函数与算术 用于操作列表的高阶折叠	B AP
4	1392	3	从布尔公式中提取变量名 基于异常的回溯与延续	B AP
5	1074	3	使用延续查找树深度和子树 用于算术表达式的延续传递风格解析器	AP AP
6	1292	13	基于选项的列表遍历 使用模块 类型与抽象之间的转换	B B AP
7	409	2	列表与字符串操作的高阶函数 可变状态与记录	B B
8	499	8	惰性编程与流 流与递归序列	B AP
9	3601	3	编程语言理论、类型检查、解释器	PT
10	2748	3	编程语言理论、合一、类型推断	PT

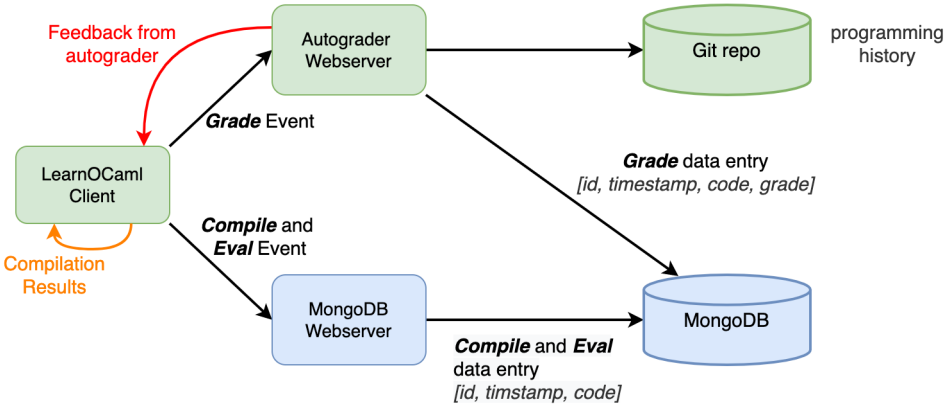
257。编程问题涵盖以下范围：1）基础技巧（标记为**B**），包括数据类型上的模式匹配、基础高阶函数（HOF）和递归；2）高级编程 技巧（标记为**AP**），例如回溯问题、延续、流和惰性编程；3）编程语言理论，涵盖编程概念，包括计算自由变量和未使用变量，以及扩展现有的类型检查器和解释器实现（标记为**PT**）。

2.2 基准测试集λ修复：修复语法、类型和逻辑错误代码（150 个问题）

我们的第二个基准测试集取样自实际存在错误的学生代码。这些学生程序收集于 2022 年秋季函数式编程课程的一轮教学中，课程使用了修改版的 LearnOCaml 平台 [5, 19]，该平台允许学生在同一处编辑、编译、测试和调试 OCaml 代码。

基于 Learn-OCaml 平台，我们构建了一套数据收集流程，可自动记录学生的操作 [8]。具体来说，我们通过异步日志请求，将编译和求值等本地编程事件（用于测试和调试）发送到后端服务器。图 1 展示了从在线环境 Learn-OCaml 收集数据的过程。

约 52.81%（即 320 名学生中的 169 名）同意我们分析他们与 Learn-OCaml 平台的交互。我们使用该流程收集了超过 270,000 条编程事件，每条事件都存储了代码快照以及



■ **图1** 数据收集流水线。所有提交数据都存储在*MongoDB* 数据库中。浅绿色组件是 Learn-OCaml 平台的原有组件，浅蓝色组件则是我们新引入的组件。

反馈信息（例如静态错误、成绩等）。我们从这些编程事件中，为以下三类各抽取了 50 个不同的有缺陷的程序：语法错误、类型错误和逻辑错误。与 λCodeGen 中的多函数求值不同，所有有缺陷的程序都是单函数程序。我们确保有缺陷的程序均匀分布在 10 份作业中，以平衡难度等级。

2.3 基准测试集λ解释：概念性问题（50道问题）

该基准测试集包含50道概念性问题，这些问题由授课教师在本课程2024年秋季和2021年秋季两次开课期间使用。其中包括作为备考材料提供给学生的问题，以及正式考试中使用的问题。这些问题考查学生对变量作用域、构造归纳证明、推导类型、程序求值和应用替换（即用一个表达式替换自由变量）的理解。作答时不得使用编译器。

■ **表2** 按主题和难度分组的概念性问题

# 问题	主题	难度
7	绑定、类型、递归函数	B
10	绑定、类型、递归函数	A
4	OCaml 基本概念	B
9	归纳法与程序推理	B
6	归纳法与程序推理	A
6	程序求值	B
8	程序求值	A

每道题都经过人工标注，注明其难度等级。与旨在培养实践技能的编程练习不同，概念性问题通常侧重于评估学生对基本原理和理论的理解，且往往需要解释、证明或进行高层次推理。

我们为每道题标注了难度等级：**B**（基础）或 **A** （高级）。表 2 展示了 50 道题的难度等级分布及其主题。

2.4 所选大语言模型概览

为了评估大语言模型在函数式编程中的表现，我们从多个维度选取了模型：来自主要提供商（Meta、OpenAI、Google、Anthropic）的领先系统，以便进行跨公司比较；闭源付费模型（Claude 3.7 Sonnet、GPT 系列、Gemini 系列）和免费开源模型（Llama 3 系列、Qwen2.5 7B）兼顾，以考虑可及性和成本，尤其是对学生而言；来自同一提供商、参数规模不同的模型（例如 7B 与 70B），以评估规模的影响；以及面向聊天和侧重推理的模型，以呈现其问题求解能力的差异。

所有大语言模型的结果均通过官方 API 调用获得，未修改采样或解码参数。除非另有说明，我们使用各 API 提供的默认设置（例如温度、最大令牌数、top-p）。尽管在实验期间默认设置保持稳定，但提供商今后的更改可能会影响结果的精确复现。

表 3 列出了我们实验中研究的 9 个大语言模型，以及它们所属的公司、模型规模（“规模”列[2]）和模型类型。

2.5 求值方法

我们通过 OpenAI 和 OpenRouter API 与所有大语言模型交互。[3] 为确保稳定性并减少大语言模型产生的幻觉，我们使用相同的基准问题提示每个模型 5 次。对于 λRepair 和 λExplain，共获得 250 个回答（5 x 50 个问题）；对于 λCodeGen，共获得 50 个回答（5 x 10 项作业）。我们采用自动评分和人工评分来评估大语言模型生成的回答。

我们的求值方法遵循两项原则：第一，我们以预期学生会采用的方式向大语言模型发出提示；第二，我们从授课教师的角度，依据正确性、算法设计和可读性等标准评估模型的回答。对于家庭作业，我们要求仅输出代码，并向模型提供一份包含多个编程任务的完整作业，这使代码生成比常见的单任务评估更具挑战性。这样，我们就能更好地衡量大语言模型在教育环境中的实用性和有效性。我们在提示中要求给出简洁的 答案或解释，以避免冗长。为支持结果的可复现性和重复使用，我们将把基准测试集以及提示脚本和说明作为制品提供 [60]。该套件

[2]部分模型的参数量未由其提供方正式披露。我们报告的是根据公开讨论的估算得出的近似规模：GPT-4o-mini ≈ 8B、GPT-4o ≈200B、ClaudeSonnet3.7 ≈ 400B、Gemini2.0 Flash ≈ 200 - 300B，以及 o3-mini —未知。这些数值来自轶闻或推测，应视为粗略估计；标注为“–”的条目表示截至撰写本文时，没有可靠的公开信息。[3]https://openrouter.ai/。访问日期：2025-02-10。

将结构清晰、文档齐全，使其他研究人员能够轻松部署，并将其应用于新模型。

■ **表3** 本次求值中纳入考量的最先进的大语言模型，按公司、参数规模和模型类型列出。

模型	公司	Size	Type
Claude 3.7 Sonnet 4	Anthropic	–	聊天/推理
GPT-4o 5	OpenAI	–	聊天/推理
GPT-4o Mini	OpenAI	–	聊天/推理
o3-mini	OpenAI	–	聊天/推理
Llama 3.1 70B Instruct 6	Meta	70B	Chat
Llama 3.1 8B 指令版	Meta	8B	Chat
Gemini 2.0 Flash 7	Google DeepMind	–	聊天/推理
Gemini 1.5 Flash 8B	Google DeepMind	8B	聊天/推理
Qwen2.5 7B [55]	阿里云	7B	基础版（通用）

基准测试集的设置与评估 λ代码生成 对于 λ代码生成，我们提供给定任务的完整自然语言作业描述以及类型规范。随后，我们首先使用编译器和自动评分器（自动评估）对每个生成的答案进行评分，然后由 2 名经验丰富的助教对生成的 50 个答案进行人工评估。在人工评估过程中，我们会检查每个解决方案，确认生成的代码是否满足问题要求。典型问题包括：1）问题要求提供尾递归函数或使用特定的高阶函数，但未提供或未使用；2）在问题描述禁止使用库函数和/或状态的情况下，使用了不允许的函数或命令式特性；3）提供了总体效率低下的算法。在当前研究中，此类人工评估通常缺失（例如，[22]）。

我们根据三个类别对答案进行评分：正确性、算法设计和可读性（见表 4）。在每个类别中，生成的答案随后会被评定为精通、熟练、发展中、初级或不可评分。要将一个解决方案评定为精通级别，它必须是正确的（即通过自动评分器的所有测试用例），并且正确遵循问题要求。如果存在上述三类问题中的任何一项，则会被归类为熟练或更低等级。

我们按照层级依赖关系对答案评分，依次评估正确性（对问题的理解） >算法设计（策略）>可读性——也就是说，正确性是评估算法设计的前提，而算法设计是评估可读性的前提。如果解决方案无法编译，则无法自动评分，因此判定为错误并记零分。之后的标准，即算法设计和可读性，也就不再适用。每个生成答案的最终等级为其在这三项标准上得分的平均值。

[4]https://www.anthropic.com/news/claude-3-7-sonnet。访问于2025-02-10。[5]
https://openai.com/index/hello-gpt-4o/。访问于2025-02-10。[6]
https://www.llama.com/models/llama-3/。访问于2025-02-10。[7]
https://deepmind.google/models/gemini/flash/。访问于2025-02-10。

表4 代码生成评分标准。包含语法错误或类型错误的解决方案将标记为不可评分。

精通（4分）	熟练（3分）	发展中（2分）	初级（1分）
正确性 该解决方案通过了全部测试用例，且没有风格问题。	该解决方案通过了至少 75% 的测试用例且没有风格问题。	The 解决方案 失败 许多测试用例（介于 75 % 和 25 % 之间通过率）。	解决方案可以编译但通过的测试用例少于 25% 的测试用例。
算法设计 The 算法 or 实现技术 匹配 the 问题规格说明。	The 算法 or 实现技术 大多 问题 符合规范。	The 算法 or 实现技术 几乎不 问题 符合规范。	The 算法 or 实现技术不连贯或不恰当（例如，uses 不允许 or 不必要 辅助 函数）。
可读性 仅生成必需的代码。	生成代码，但包含 冗余的 函数或测试用例。	生成代码，但其包含过于冗长或不必要的内容自然语言解释。	不适用（如果解决方案被评为初级在正确性和算法设签名，它也被评为初级（就可读性而言）。

基准测试集的设置与评估 λ修复 对于 λ修复，我们向大语言模型提供一个有缺陷的程序及其相应的错误消息，并要求大语言模型在没有先前示例的情况下生成修复后的版本，即在零样本 设置下提示模型。请注意，我们使用单错误缺陷程序测试模型，因为我们的编译器只会识别程序中的第一个 错误。评估 λRepairSyntax和 λRepairType的答案时，我们会将语法错误/类型错误以及 OCaml 编译器生成的错误消息连同有缺陷的程序一起提供给大语言模型。评估 λRepairProg的答案时，我们会将有缺陷的程序与单函数 作业描述（即作业中对应于该单个函数的问题）一起提供给大语言模型。

我们省略了所有预期、通过或未通过的测试用例。我们以与λCodeGen 问题相同的方式，使用自动评估和人工评估来评定生成的答案（见表 5）。

基准测试集的设置与评估 λ解释 最后，对于 λ解释，我们用自然语言考试题提示模型。对于需要结构化回答的问题（例如状态图），我们提供一个示例来引导模型，以确保其缺乏先前的格式知识不会影响表现。我们调整了编程问题的评分量表，使其更适合表 6 中的概念性问题。具体而言，我们考虑对问题的理解、解题所用的策略以及对解释的表达。我们的

表5 代码修复评分标准。包含未修改代码、非 OCaml 代码、无意义文本或不含代码的回答将被评为不可评分。

精通（4 分）	熟练（3 分）	发展中（2 分）	初级（1 分）
正确性 完美修复代码（即能够通过编译器的代码）。	修复代码，使其通过编译器，但会造成冗余更改（例如，添加额外的辅助函数或测试用例），或者不必要地重新格式化代码。	正确定位了错误，但未能重新配对代码，或引入一种新的同类型错误。	未能定位该错——错或修复程序。

表6 概念性问题／非编程问题的评分标准。离题的解决方案或包含无意义文本的答案将被标记为不可评分。

精通（4分）	熟练（3分）	发展中（2分）	初级（1 分）
对问题的理解 展现出对问题的深入理解，识别出问题中的所有关键组成部分and 潜力 挑战。	对问题有很好的理解，能够识别大部分问题，指出多数关键组成部分和挑战。	展示了对问题的基础standing of the problem but may miss一些关键组成部分或误解了某些某些方面。	难以理解问题及其要求，导致解答中出现严重的错误。tion。
策略 阐述 a 清除以正确的方式解决问题。	仅阐述了恰当策略的部分内容，但大多数关键要素均已包含。	阐述了一项策略但仅部分有用；关键要素可能缺失或不提供了适当的步骤。	未能提供策略，以应对问题。
可读性 提供清晰、完整、 明确无歧义、且简洁的解释性。	提供完整的解释，但可能在某些方面不够清晰或简洁，度。	提供了含糊不清的，不一致，或不完整的解释；包含不需要的信息。信息。	提供了不连贯的解释 or未能给出一个。

评分再次采用层级依赖：理解 > 策略 > 表达。缺乏正确理解的回答不可能拥有有意义的策略，而薄弱的策略会削弱有效表达。

2.6 大语言模型的总体字母等级

使用上一节所述的掌握度评分方法，我们为大语言模型生成的每个解决方案指定一个精通、熟练、发展中、初级或不可评分级别。对于每个大语言模型，我们接着计算在给定级别上解决的问题所占的总体百分比：#(该级别的问题数) / #(问题总数)。为了

更好地对大语言模型进行分类，我们还会根据大学的成绩换算政策（见表7），将掌握水平转换为字母等级，从而为大语言模型计算字母等级。大语言模型的总成绩是所有等级的平均值。例如，若某个大语言模型的结果为：80 % 精通、10 % 熟练和 10 % 发展中，则其最终字母等级为 A-（8.0）。

表7 计算大语言模型字母等级的成绩转换政策。

精通级别	字母等级	中点成绩	成绩区间
精通	A	8.45	8.2-8.7
	A-	7.95	7.7-8.2
	B+	7.45	7.2-7.7
熟练	B	6.95	6.7-7.2
	B-	6.45	6.2-6.7
	C+	5.95	5.7-6.2
发展中	C	5.45	5.2-5.7
初级	D	4.95	4.7-5.2
N/A	F	0.0	0.0

3 结果

现在，我们重新审视引言中提出的研究问题。

对于所有这些问题，我们的求值结果显示，各模型在所有任务中的表现明显分为几个层级。o3-mini、Claude 3.7 Sonnet 和 GPT-4o 始终位居前三。相反，Gemini 1.5 Flash 8B、Qwen2.5 7B、Llama3.1 8B，以及偶尔出现的 GPT-4o-mini，则始终排名靠后。因此，我们在表格中将这十个模型划分为三个层级，以便于比较。

3.1 RQ1：大语言模型能否正确、简洁地完成函数式编程作业 ?

我们对九个大语言模型在 λCodeGen 上的评估（表8）显示，这些模型之间存在显著的性能差距，即加权总成绩 / 掌握率（%）的标准差分别为 15.8 / 20.6。排名前三的模型——o3-mini、Claude 3.7 Sonnet 和 GPT-4o——展现出强大的能力，掌握率约为 70%，加权总成绩达到 B 等级。相比之下，Qwen2.5 7B 和 Llama3.1 8B 等较小的封闭资源模型表现明显较差，分别为 21.9% 和 15.8%，因此等级为 F。除顶尖模型外，本研究中 40% 的大语言模型（排名靠后的 4 个大语言模型）只能为不到 41% 的问题生成达到精通级别的解决方案。平均而言，大语言模型能够以精通级别解决约一半（52.6%）的编程作业题（见表8）。这仍然是一个令人警醒的结果。

在 λCodeGen上，排名前三的模型（GPT-4o、o3-mini和Claude 3.7 Sonnet）相当有效，因为它们分别在69.8%、74%和71.3%的问题上达到了精通级解决方案。这些成功率低于在高资源Python/Java基准测试（如HumanEval [9] 和MBPP [3], ）上所报告的结果，其中

表8 基准测试集 λ代码生成生成的代码评估（M：＝精通；P：＝熟练；D：＝发展中；B：＝初级；N：＝不可评分）绿色高亮行表示商业（付费）模型的结果，其余行则表示开源（免费）模型的结果

模型	M（%）	P（%）	D（%）	B(%)	N（%）	等级	平均等级评分
o3-mini	73.6	3.4	2.3	12.1	8.7	B+	
Claude 3.7 Sonnet	71.3	3.8	4.5	8.3	12.1	B	B
GPT-4o	69.8	3.0	3.0	12.5	11.7	B	
Gemini 2.0 Flash	58.9	1.5	2.6	9.1	27.9	C+	C+
Llama3.3 70B	58.5	2.3	1.9	10.9	30.9	C+	
Gemini 1.5 Flash 8B	40.8	0.8	2.6	13.2	42.6	D	
GPT-4o-mini	40.4	0.0	6.4	24.9	28.3	D	F
Qwen2.5 7B	21.9	0.8	4.2	22.3	50.9	F	
Llama3.1 8B	15.8	0.0	4.2	12.1	67.9	F	

平均成绩	成绩中位数	成绩标准差	M 平均值（%）	M 中位数（%）	M 标准差（%）
54.70	58.0	15.8	52.6	58.7	20.6

顶尖模型的准确率超过90%。尽管如此，与SWE-Bench [22]等大型真实世界程序相比，大语言模型在λCodeGen上仍然有效。与早期一代在低资源语言上的表现相比，我们的结果优于2022年的Codex（通过率约40%） [7] 以及使用知识迁移的Llama [6], ，这表明如今的通用大语言模型能够在低资源语言中取得强劲结果，并且模型的持续演进可能会进一步缩小与高资源环境之间的差距。

与专用代码合成工具相比， [35, 39],大语言模型提供了不同的权衡：领域专用工具需要精确的输入和类型签名，并能为小型函数式编程任务（15–30 个抽象语法树节点）保证正确的解决方案，而大语言模型能够处理更大、更开放的任务，代价是偶尔出现幻觉。为扩展这一比较，我们运行 BURST [35], 一种基于自底向上搜索与天使式执行的先进 OCaml 合成器，在单函数设置下针对 λCodeGen 进行实验。

我们的结果表明，BURST 仅能合成 11.3% 的问题，主要是那些涉及结构递归的问题，例如树遍历操作。尽管 BURST 在为具有精确规范的递归函数可靠地生成正确的类型签名和实现方面表现出特别的优势，但其局限性在 CPS（延续传递风格）、高阶函数、惰性或无限流、记忆化、基于异常的回溯以及多步推理任务上十分明显。然而，当它成功时，它能可靠地生成正确的类型签名，并在简单递归函数（例如树遍历）上表现良好。这一对比说明了两种方法各自的作用：专用工具在更狭窄、定义明确的领域内提供有保证的正确性，而大语言模型能够根据自然语言提示自适应地生成代码，这也解释了新手为何可能选择使用大语言模型而非领域专用工具。

排名最低的四个大语言模型经常生成被归类为“不可评分”的代码，其中生成的代码甚至无法编译（Llama3.1 8B 为 67.9%，Qwen2.5 7B 为 50.9%，Gemini 1.5 Flash 8B 为 42.6%）。无法编译最常见的原因是语法和类型错误，此外还包括无效代码结构（占 Llama 3.1 8B 响应的 6.1%）以及错误的注释格式（占 Qwen2.5 7B 响应的 7.5%）。Llama 系列模型还特有地生成了一些毫无意义的文本作为响应。

我们还注意到，在所有大语言模型中，被归类为初级的响应比例高于被归类为熟练或发展中的响应比例。我们的人工评估显示，被归类为初级的响应通常能够通过类型检查，但无法满足给定规范、包含逻辑错误，或只能通过少数甚至无法通过任何测试用例。即使是 GPT-4o 这样的顶尖模型，也偶尔无法遵循提示词，例如在作业 7 中使用了明确禁止的高阶函数；反之，o3-mini 在作业 3 中创建了不必要的辅助函数，而没有直接使用要求的高阶函数。这似乎表明，大语言模型未必理解所给出的完整规范。尽管 GPT-4o 是函数式编程作业中表现最好的大语言模型之一，但其 69.8% 的精通率表明仍有很大的改进空间。这些结果表明，总体而言，大语言模型尚不足够可靠，无法有效解决函数式编程任务，但对新手具有很强的适应性。

3.2 研究问题 2：大语言模型能否正确且简洁地修复函数式程序中的语法、类型和逻辑错误?

本文评估大语言模型在实际学生程序中进行错误修复的效果——从语法和类型错误到逻辑错误。

研究问题 2a：大语言模型能否正确且简洁地修复语法错误? 表 9 中的结果显示，在语法错误修复方面，排名前三的大语言模型（o3-mini、GPT-4o 和 Claude 3.7 Sonnet）在我们的示例中达到精通级别的比例均超过 78%。综合所有大语言模型，我们观察到明显的梯度式下降：GPT-4o-mini（67%）和 Gemini 2.0 Flash（62%）处于中间梯队，而开源模型（Llama3.3 70B、Qwen2.5 7B）的精通比例为 42%–59%。值得注意的是，大多数大语言模型（10 个中有 6 个）都成功修复了至少 60% 的有缺陷的程序。对于较低梯队的模型，我们还观察到，与代码生成相比，它们在代码修复方面的表现有明显提升。

表 9 对基准测试集 λ语法修复生成代码的求值（M：= 精通；P：= 熟练；D：= 发展中；B：= 初级；N：= 不可评分）绿色高亮行表示商业（付费）模型的结果，其余则为开源（免费）模型的结果

模型	M（%）	P（%）	D（%）	B（%）	N（%）	等级	平均等级评分
o3-mini	82.4	3.2	3.2	1.6	9.6	A-	
GPT-4o	81.2	8.4	6.4	4.0	0.0	A-	A-
Claude 3.7 Sonnet	78.4	5.2	14.0	2.4	0.0	A-	
GPT-4o-mini	67.2	2.4	24.4	6.0	0.0	B+	
Gemini 2.0 Flash	62.0	8.4	19.2	6.4	4.0	B+	B+
Llama3.3 70B	58.8	2.4	24.4	6.4	8.0	B	
Qwen2.5 7B	47.6	2.4	44.4	2.4	3.2	B	
Gemini 1.5 Flash 8B	43.6	5.2	40.0	5.4	5.8	B	B
Llama3.1 8B	42.4	8.4	30.4	5.6	13.2	B-	

平均成绩	成绩中位数	成绩标准差	M 平均值（%）	M 中位数（%）	M 标准差（%）
B+	B+	4.96	64.7	64.6	15.6

表现最佳的大语言模型超过或不逊于专用工具，例如 SYN- SHINE；该工具在小型（少于 100

词元）Java 错误程序上的准确率达到 82% [1]。此外，RING 是一种由 Codex 和少样本学习驱动的语法错误修复工具 [23]，在 Python 上的准确率达到 94%。这表明，基于示例的提示可能会提升结果，因此我们在后文采用了单样本学习。

总体而言，我们的结果进一步证实，大语言模型正逐渐具备与专用语法修复工具相抗衡的能力，同时还提供了更大的灵活性。

RQ2b：大语言模型能否正确、简洁地修复类型错误？语法错误纠正取得了显著成果，而类型错误则面临更具挑战性的情形（表10）。排名前三的商业模型（o3-mini、GPT-4o、Claude 3.7 Sonnet）仍取得了令人瞩目的掌握率（72-83%）。然而，与语法错误修复（λRepairSyntax 基准测试集）相比，开源模型的表现（Llama3.3 70B：60.4%，Llama3.1 8B：40.8%）下降得更为明显。这表明这些大语言模型并未接受过推理问题方面的训练。

表10 基准测试集 λRepairType生成代码的求值（M：= 精通；P：= 熟练；D：= 发展中；B：= 初级；N：= 不可评分）绿色高亮行表示商业（付费）模型的结果，其余行则表示开源（免费）模型的结果

模型	M（%）	P（%）	D（%）	B（%）	N（%）	等级	平均等级评分
o3-mini	80.2	5.6	11.2	3.0	0.0	A-	
GPT-4o	75.6	3.2	16.4	6.2	0.0	A-	A-
Claude 3.7 Sonnet	72.0	2.4	20.4	5.2	0.0	A-	
Llama3.3 70B	60.4	1.6	18.6	11.4	8.0	B+	
GPT-4o-mini	57.6	3.0	26.4	11.0	2.0	B+	B+
Gemini 2.0 Flash	55.6	4.0	22.4	12.0	6.0	B+	
Qwen2.5 7B	42.4	4.0	36.4	13.0	4.0	B	
Llama3.1 8B	40.8	3.0	25.0	17.2	14.0	B	B
Gemini 1.5 Flash 8B	37.6	2.0	41.4	14.0	5.0	B-	

平均成绩	成绩中位数	成绩标准差	M 平均值（%）	M 中位数（%）	M 标准差（%）
B+	B+	4.96	60.5	59.0	15.9

将大语言模型与专用类型错误纠正工具进行比较，难分高下。一方面，大语言模型的表现优于 RITE 等 OCaml 专用系统；这些系统也会使用学生的有缺陷代码，而 MENTAT [44, 45], 报告称，在大规模 OCaml 基准测试中，成功率仅为 39 %/45 %。不过，这些系统在我们基准测试中的小型程序上可能更高效，因此值得在未来的研究中进行比较。另一方面，大语言模型的表现不及 Python 专用工具 PyTy，其准确率达到了 85.4 % [10]。近期关于 Haskell 的研究[46] 展示了另一种互补方案：将 GPT-4 与基于编译器的故障定位和 QuickCheck 验证相结合，修复成功率可达 88.5 %。

总体而言，这些结果表明，通用大语言模型在精确性方面可能无法与针对特定语言开发的专用类型错误纠正工具相匹配。与此同时，大语言模型展现出了出人意料的强大跨语言能力，即使面对 OCaml 的复杂类型系统也是如此；鉴于目前针对该系统的专用工具寥寥无几，大语言模型因而可能成为初学者学习新语言时的宝贵资源。大语言模型在单函数语法和类型错误修复方面表现尤为出色，这可能是因为这些

任务能够隔离目标错误，只需进行局部修复；相比之下，多任务代码生成需要复杂的推理能力。

RQ2c：大语言模型能否正确、简洁地修复逻辑错误？ 排名前三的大语言模型（o3-mini、Claude 3.7 Sonnet 和 GPT-4o）掌握率均超过 60%（表11）。这表明它们在逻辑错误修复方面的能力弱于语法或类型错误修复，但这些模型始终能够生成完全正确的解决方案。中游模型包括 GPT-4o-mini 和 Llama3.3 70B 等，它们取得了一定成效，但部分正确或存在错误的回答比例更高。排名靠后的模型（Gemini 1.5 Flash 和 Qwen2.5）表现明显较差，掌握率低于 45%。

表11基准测试集 λRepairProg生成代码的求值。（M：= 精通；P：=熟练；D：= 发展中；B：= 初级；N：= 不可评分）绿色高亮的行表示商业（付费）模型的结果，其余则为开源（免费）模型的结果

模型	M（%）	P（%）	D（%）	B（%）	N（%）	等级	平均等级评分
o3-mini	72.4	5.1	6.0	6.5	10.0	A-	
Claude 3.7 Sonnet	68.0	6.0	3.0	1.0	15.0	A-	A-
GPT-4o	67.2	3.6	1.6	7.0	20.6	A-	
Llama3.3 70B	56.2	1.6	5.2	16.0	21.0	C+	
GPT-4o-mini	52.8	4.0	11.0	12.2	20.0	B+	B
Gemini 2.0 Flash	48.8	3.6	8.0	18.6	21.0	B	
Qwen2.5 7B	43.4	4.0	5.0	19.6	24.0	C	
Llama3.1 8B	37.6	3.0	7.0	29.4	20.0	B	B-
Gemini 1.5 Flash 8B	33.6	3.0	5.0	39.0	24.0	B	

平均成绩	成绩中位数	成绩标准差	M 平均值（%）	M 中位数（%）	M 标准差（%）
B+	B	6.52	55.5	54.5	14.31

以往用于教育的 APR（自动化程序修复）工具，如用于 Python 的 CLARA 和 Refactory（[18, 21]），也依赖已有的正确提交或测试用例来修复学生作业中的错误程序。这些系统旨在教育环境中提供可扩展的实时反馈。尽管 Refactory 和 CLARA 在各自的 Python 基准测试集上报告的修复成功率超过 90%，顶尖大语言模型也能达到相当高的掌握率，而且易于在 OCaml 等不同语言中重复使用。

使用单样本学习进行代码修复的进一步研究除了零样本求值外，我们还评估了 λRepairProg在修复语法和类型错误方面的单样本学习能力。单样本学习是指向模型提供一个任务示例（一次演示），随后要求它对新的输入执行该任务。它不同于展示多个示例的少样本学习，也不同于不展示任何示例的零样本学习。

这一设置使我们能够评估少量上下文引导是否能提升代码修复表现。我们重点考察学生广泛使用的 GPT-4o，以及开源模型 Llama 70B。如表 12 所示，对于这两种模型，单样本学习的表现始终优于零样本提示，尽管总体提升幅度较小。GPT-4o 在语法错误和类型错误方面的精通度分别提升了 4.4% 和 2.2%，

同时初级水平的回答有所减少。Llama 70B 的提升略大，不可评分的输出最多减少了 6.6%，这与较弱模型能从上下文指导中获益更多的直觉相符。鉴于样本量相对较小，仅包含 50 个有缺陷的程序，这些结果应被视为提示性趋势，而非确凿的改进证据；但它们表明，即使是少量上下文示例，也有望引导大语言模型取得更好的修复表现。

表12 从零样本提示到单样本提示，GPT-4o 与 Llama-70B 的性能提升（百分点变化）。此处 0.0 表示无变化。

错误类型	模型	Δ 精通	Δ 熟练	Δ 初级	Δ 不可评分
语法	GPT-4o	4.4	2.0	−6.4	0.0
	Llama-70B	5.6	3.8	−2.8	−6.6
Type	GPT-4o	2.2	2.0	−4.2	0.0
	Llama-70B	3.6	4.0	−2.0	−5.6

3.3 RQ4：大语言模型生成正确且简洁答案的能力是否会因问题类型而异——代码生成、代码修复还是解释理论概念？

表 13 λ解释长答案的求值。（M：= 精通；P：= 熟练；D：= 发展中；B：= 初级；N：= 不可评分）绿色高亮行表示商业（付费）模型的结果，其余行则表示开源（免费）模型的结果

模型	M（%）	P（%）	D（%）	B（%）	N（%）	等级	平均等级评分
o3-mini	80.0	15.2	4.8	0.0	0.0	A	A-
Claude 3.7 Sonnet	60.0	18.4	22.8	0.4	0.4	A-	
GPT-4o	58.4	13.6	23.2	4.8	0.0	B+	
Gemini 2.0 Flash	44.4	10.8	21.6	15.2	8.0	B+	B
GPT-4o-mini	48.8	9.2	20.4	20.0	1.6	B+	
Llama3.3 70B	42.8	2.4	28.4	12.4	14.0	B-	
Gemini 1.5 Flash 8B	36.8	16.8	30.4	13.6	2.4	B	B-
Qwen2.5 7B	34.8	21.6	22.4	17.6	3.6	B-	
Llama3.1 8B	16.8	9.6	12.4	49.2	12.0	B-	

平均成绩	成绩中位数	成绩标准差	M 平均值（%）	M 中位数（%）	M 标准差（%）
C	C+	15.8	50.4	46.6	19.1

最后，我们评估模型解释编程语言设计基本概念的能力。从学生的角度来看，这是一个特别重要的领域，因为它帮助学生学习材料，并获得授课教师、助教或课程笔记所提供内容之外的额外解释。这也是学生更难评估正确性的领域，因为通常没有简便的自动化方法可用于测试所生成的回答。

表 13 中的结果显示，在生成解释以及回答 λExplain 基准测试集中的问题方面，评估的 9 个大语言模型能力差异显著。值得注意的是，o3-mini 仍表现出色，获得了

总体 A 等级。其掌握率进一步表明，它能够在超过 80% 的情况下提供正确且简洁的解释。

然而，大多数模型表现不佳。中等等级和较低等级模型的加权平均成绩分别为 B 和 B-，这表明大语言模型解释理论概念的能力存在显著差距。与 λ代码生成相比， λ解释的整体表现似乎更好，主要原因在于不可评分率显著更低。然而，这种表现具有误导性，因为正如熟练、发展中和初级的比例所显示的那样，许多回答仍然包含不准确之处或表述不清的解释。这一点尤其令人担忧，因为学生通常无法验证或测试生成的回答。我们的结果与现有研究一致，包括 [24]，他们观察到 ChatGPT 在 StackOverflow 上的回答有一半以上包含错误。同样，Sarsa、Denny、Hellas 和 Leinonen [47] 发现，Codex 有 32.8% 的回答包含错误解释。

一个常见问题是不必要的冗长，尽管我们在提示中明确要求提供简洁的解释。例如，当被要求提供一个尾递归 OCaml 程序时，Claude Sonnet 和 Gemini Flash 等模型通常会同时给出尾递归和非尾递归版本，随后又长篇解释尾递归的重要性。这些回答被评为熟练，尽管它们正确回答了主要问题。然而，回答过于冗长会分散人们对答案核心内容的注意力，而且生成的答案略微偏离主题。

3.4 RQ4：大语言模型生成正确且简洁答案的能力是否会因不同问题类型而异——从代码生成到代码修复，再到解释理论概念？

为回答这一问题，我们比较了为每个大语言模型在给定任务上评定的总体字母等级（见图 2）。这表明大语言模型在三项任务上的表现存在明显差异。我们观察到，所有大语言模型在代码生成问题（ λCodeGen）上始终最为吃力。在该基准测试集上，即使表现最好的模型也仅获得 B 成绩，而较小的免费模型（如 Gemini 1.5 Flash 8B 和 Llama 3.1 8B）得分更差（D 或更低）。我们的比较还表明，大语言模型各有所长。虽然 o1、o3-mini，以及 Gemini 2.0 Flash，还有出人意料的 Llama 3.1 8B 等模型在 λ解释基准测试集上得分最高，但所有其他大语言模型在来自 λRepair语法的问题上得分最高。此外，我们观察到，对于若干模型（Qwen 2.5 7B、Llama 3.1 8B、GPT-4o Mini 和 Gemini 1.5 Flash 8B），其生成代码的能力（ λCodeGen）与其他任务之间存在高达 3-4 个成绩的巨大差距。对于表现顶尖的模型，不同任务之间的这一性能差距则窄得多。

3.5 研究问题 5：问题的难度等级如何影响大语言模型在不同问题类型下的回答？

我们还通过统计每个难度等级的问题数量，并分别除以 λCodeGen 和 λExplain 中基础/高级问题的总数，计算基础/高级问题的掌握百分比。结果见图 3。对于代码生成，

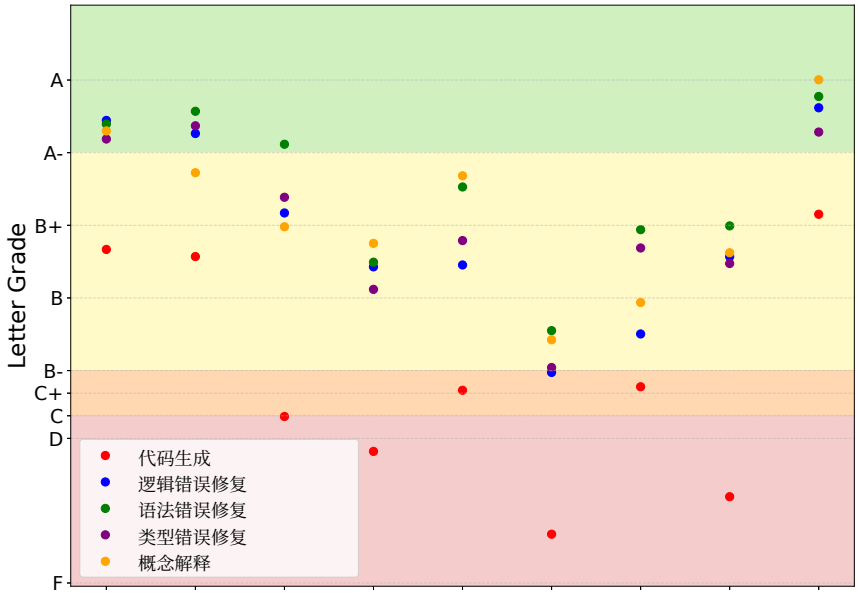


图 2 大语言模型在不同任务中的加权成绩比较——涵盖代码生成、代码修复和理论概念解释

我们观察到，所有大语言模型在基础问题上的表现通常都优于高级编程问题，不过差距并不大。表现最好的模型（Claude 3.7 Sonnet、GPT-4o、Gemini 2.0 Flash 和 o3-mini）在两个难度等级上的掌握率均超过 60%。这表明，它们不仅能够处理简单任务，也能较好地运用更高级的编程技术。

基础任务，例如模式匹配、简单的尾递归和高阶函数，完成率较高，尤其是在问题简短且定义清晰时。高级任务，包括递归、延续、惰性编程或规模较大的代码生成问题，则会导致更多类型错误。这表明，大语言模型能够为较简单的任务生成正确代码，但在处理复杂的 OCaml 程序时，缺乏更深入的类型推理能力。

编程理论（**PT**）问题的差距更为明显。基础问题的掌握率几乎是**PT** 问题的两到三倍，而 Llama 3.1 8B 和 Qwen2.5 7B 等较弱模型的掌握率为 0%。这些理论问题涉及实现解析器和一种类似 OCaml 的编程语言，并考查自由变量、替换、求值和类型推断等概念。表现不佳凸显了一个关键局限：虽然大语言模型能够为编程问题生成可运行的代码，但面对需要深入理解编程语言理论和进行精确类型推理的任务时，它们仍然力有不逮。这在意料之中，因为当前模型可能只是从训练示例中进行模式匹配，而非真正理解底层的计算理论。

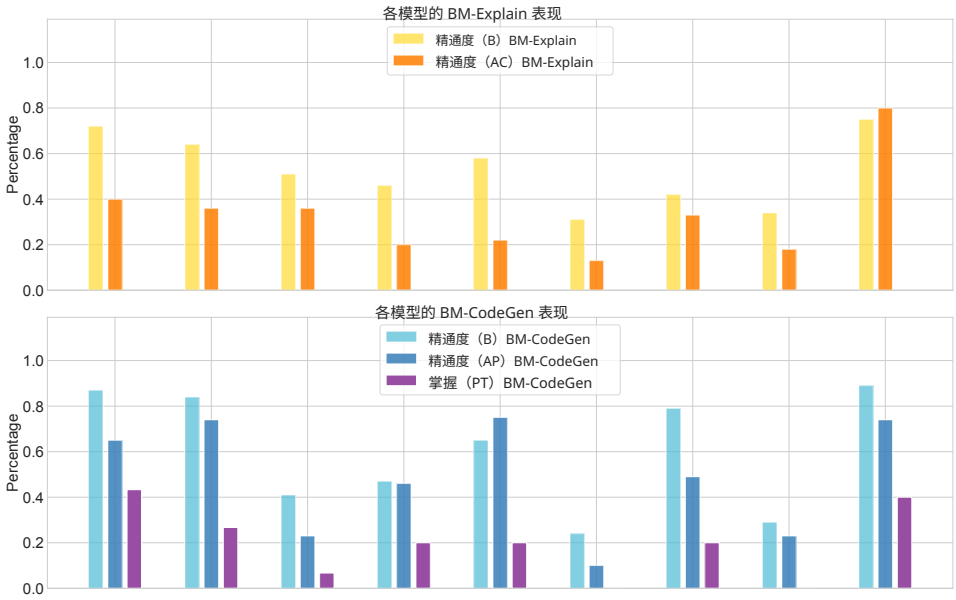


图 3 各模型在**B**（基础）、**AC**（高级概念）/**AP**（高级编程）和**PT**（编程技术）问题上的掌握率。BM-CODEGEN 表示 λCodeGen，而 BM-Explain 表示 λExplain。

在答案生成（ λ解释）方面，难度趋势相似：基础问题更容易；对大多数模型而言，高级（**AC**）问题的掌握率约为基础（**B**）问题的一半。GPT-4o 和 o3-mini 等顶尖推理模型在**AC** 问题上掌握率更高，但面对类型推断、替换和动态求值等复杂主题时仍然吃力。例如，它们往往能给出正确的中间推理，却无法将各步骤整合成准确的最终答案。变量作用域问题（遮蔽和覆写）常常导致解释和示例有误。相比之下，这些模型能够很好地处理尾递归分析，尽管代码实现中存在风格问题；这意味着它们可能具备分析尾递归函数的推理能力，却难以在代码中实现这一概念。

4 效度威胁

本研究的一项局限在于求值过程可能引入偏差。例如，将生成的回答人工划分为精通、熟练、发展中、初级和不可评分等类别，可能带有主观性。采用多位评分者有助于减轻这一问题，并提高分类的可靠性。在 λRepair 基准测试集中，我们评估模型是否解决了原始错误类型（语法、类型或逻辑），但不会对修复过程中引入的

新错误进行扣分。这符合修复工具的标准做法，即针对特定错误类型，从而能够比较模型在不同错误类别上的表现。不过，同时追踪初始正确性和最终正确性，或许能更全面地衡量模型的有效性。此外，在其他竞争性修复工具上运行 λRepair，也是未来值得探索的方向。再者，在修复逻辑错误时，研究向大语言模型提供了原始问题描述以及有缺陷的代码。然而，目前尚不清楚模型是否真的尝试修复给定的逻辑错误，还是仅仅从头生成一个新的问题解决方案。虽然可以通过只提供测试用例来检验这一点，但我们的设置旨在模拟学生的实际体验，而学生通常无法获得这类测试用例。这一局限表明，仍有必要开展更深入的研究，并将其列为未来工作的一个方向。

另一个需要考虑的问题是，我们的评估数据可能与模型的训练集有所重叠。由于我们的数据来自2022年秋季，这些数据可能已在训练过程中被模型接触过。尽管使用公开数据或课堂数据的研究普遍存在这一局限，但性能评估仍有意义，因为大语言模型即使面对熟悉的任务也可能出错。收集更新的数据集需要重新征得学生同意，因此留待未来工作开展。我们还假设大语言模型已完成训练，且在评估期间不会继续学习，这对研究的有效性和可复现性至关重要。此外，大规模评估大语言模型的成本和可及性也引发了公平性问题，因为并非所有研究人员或教育工作者都有资源复现本研究。出于同样的原因，我们的评估并未涵盖所有可用模型：我们排除了o1模型，因为其成本高昂，使得大规模复现难以实现。初步实验表明，o1的性能与o3相当，因此我们保留了价格更实惠的o3作为代表模型。因此，我们的结果应被视为具有参考意义，而非全面详尽。

5 相关工作

本节主要介绍大语言模型在软件工程领域的相关工作，以及对大语言模型的评估。在软件工程（SE）领域，大语言模型已应用于一系列任务，展现出显著潜力。这些任务包括代码生成 [4, 11, 13, 54]，程序修复 [14, 26, 43]以及为代码或理论提供解释 [34, 36]。这些广泛的软件工程应用源于其在大量代码和文本数据上的扎实训练，从而增强了其语言理解和代码理解能力。

5.1 用于代码生成的大语言模型

现有研究已广泛考察了大语言模型（LLMs）在代码生成 [4, 11, 13, 32, 52, 54, 58]和答案解释 [27, 28,34]方面的能力。然而，这些工作主要侧重于使用原始准确率、执行率 [4]，等自动化指标，或pass@k [13, 52]，等流行的求值指标来评估正确性，且往往依赖自动评分器或测试用例 [11]。

相比之下，我们的研究超越了正确性，通过纳入人工评分，从正确性、算法设计和可读性等方面评估大语言模型生成的代码的质量。这种方法使我们能够评估大语言模型能否作为可靠资源，在学生学习一门新的编程语言并修读一门中级课程时提供帮助。我们的目标与Yetistiren、Ozsoy和Tuzun [58] 的工作相似，作者通过评估代码效率和代码有效性来衡量Copilot生成的代码的质量，从而提供了比许多现有研究更为全面的分析。

在评估大语言模型时，我们的 λCodeGen 基准测试涉及多任务函数生成。这与 HumanEval [9], MBPP [3], 和 HumanEval+[31], 等基准测试不同，后者使用较短的代码片段（例如，HumanEval 的解决方案平均有 11.5 行和 24.4 个词元）。虽然大语言模型在这些任务上表现出色，但这些任务无法体现现实中的编程挑战。一些研究通过评估规模更大的代码生成任务[13]，（如竞赛级代码 [22], 或 GitHub 仓库代码 [22, 50, 52], ）来解决这一问题，结果发现模型在更复杂的问题上表现较差。然而，这些评估主要针对 Python 等高资源语言；我们的研究将探讨大语言模型的成功能否延伸至教育背景下的函数式编程。

5.2 大语言模型用于代码修复

除代码生成外，大语言模型也被广泛用于修复有缺陷的程序。先前的工作探索了各种生成式程序修复范式，包括利用错误代码的零样本方法：既有附带说明的方式 [14, 26, 43] ，也有不提供明确指导的方式 [16]。在零样本设置下，模型需要执行任务，而此前并未见过任何正确输入—输出对的示例。这种方法旨在反映学生在不依赖预先准备的示例或演示的情况下，可能如何使用大语言模型调试代码。

对语法有效的程序中语义或逻辑错误的修复受到了特别关注，相关研究通过微调 [53, 59] 或少样本学习 [38, 51] 证明了性能提升。近期工作主要聚焦于 ChatGPT/GPT-4o [48, 49, 56]。然而，这些研究关注的是主流语言，而我们的研究旨在了解其对低资源编程语言的有效性。针对语法和类型错误，已出现专门方法：[1]开发了 SYNSHINE，这是一个基于 RoBERTa、利用编译器诊断信息进行微调的系统，而 RING [23] 则利用 Codex 与少样本学习进行多语言语法修复。然而，这些基于大语言模型的工具与传统程序修复系统之间仍然存在性能差距。

类型错误修复方面已经取得了显著进展，尤其是在 Python 领域。 [10] 推出了 PyTy，这是一种基于 T5 的模型，在 2,766 个错误修复对上进行了微调；相比之下，TypeFix[40] 采用了零样本的基于提示的方法。针对 OCaml，Ribeiro、Macedo、Tsushima、Abreu 和 Saraiva [44] 提出了 MENTAT，将静态分析与 GPT-3 的生成能力相结合。尽管这些工具展现了进展，但我们的工作首次针对 OCaml 的三类错误（逻辑错误、语法错误和类型错误），对现代大语言模型进行了全面比较，并揭示了函数式语言修复所面临的独特挑战。

5.3 用于概念解释的大语言模型

以往研究考察了大语言模型在各种教育场景中的解释能力。FLIP 和 Edison 等人工智能工具 [34, 36]以及其他工作 [33]将大语言模型生成的内容整合到课堂中，学生普遍认为这些内容很有帮助。此外， [37] 使用 CodeLlama，依据与我们类似的评分标准对学生提交作业进行等级评定。基于这一基础，近期研究已经建立了用于评估计算机科学领域大语言模型解释质量的框架。Kabir、Udo-Imeh、Kou 和 Zhang [24]对 Stack Overflow 上的 517 个编程问题进行了分析，便体现了这一点。他们的研究发现，ChatGPT回答因其内容全面、表达清晰，在 35% 的情况下更受青睐，但令人担忧的是，其中 52% 包含错误信息。Ali、Rao、Mai 和 Xie [2] 开发了一种自动化基准测试方法，使用 SCS1 和 BDSI 概念清单对 10 个大语言模型进行评估。其求值结果表明，尽管大语言模型在基础编程概念方面表现良好，但在嵌套条件语句等更复杂的主题，以及较长的问题和运行时分析方面存在困难。这些局限性与我们的发现一致：大语言模型在 λExplain 基准测试集上的掌握率也更高。

Lekshmi-Narayanan、Oli、Chapagain、Hassany、Banjade、Brusilovsky 和 Rus[29] 通过比较研究进一步考察了大语言模型解释的质量。他们发现，尽管 ChatGPT 的解释与专家解答的语义相似度尚可，但其可读性较低，给新手学习者带来了挑战——这一趋势与 [24] 中的发现一致：ChatGPT 有 77% 的回答过于冗长。我们在对 λExplain 基准测试集的求值中也观察到了类似趋势，解释往往过于冗长。

6 结论

本研究全面调查了大语言模型在中级函数式编程课程中的能力。为开展这项求值，我们构建了三个基准测试集： λCodeGen、 λRepair 和 λExplain，并在函数式编程课程环境中，对 9 个最先进的大语言模型在代码生成、代码修复和概念解释方面进行了全面求值。结果表明，排名前三的大语言模型在所有任务中都非常有效。然而，大语言模型仍远非完美，表现最好的模型总体等级为 B+/B。本研究展现了当前大语言模型在函数式编程领域的概况，也为持续评估提供了框架，助力这些 AI 工具不断发展。研究为不同受众提供了见解，并推动了未来多个方向的研究。

参考文献

[1] Toufique Ahmed、Noah RoseLedesma 和 Premkumar Devanbu。“SynShine：改进语法错误修复”。载于：*IEEE* 软件工程汇刊49.4（2023年4月），第2169–2181 页。ISSN：0098-5589。DOI：10.1109/TSE.2022.3212635。

[2] Murtaza Ali、Prerna Rao、Yifan Mai 和 Benjamin Xie。“利用基准测试基础设施评估大语言模型在计算机科学概念库上的性能：挑战、机遇与批判性评估”。载于：*2024*年 *ACM*国际计算教育研究会议论文集——第*1*卷。ICER ’24。澳大利亚维多利亚州墨尔本：计算机协会，2024年，第 452–468 页。ISBN：979-8-4007-0475-8。DOI：10.1145/3632620.3671097。[3] Jacob Austin、Augustus Odena、Maxwell Nye、Maarten Bosma、Henryk Michalewski、David Dohan、Ellen Jiang、Carrie Cai、Michael Terry、Quoc Le 和 Charles Sutton。利用大语言模型进行程序合成。2021年。arXiv：2108.07732 [cs.PL]。网址：https://arxiv.org/abs/2108.07732 （访问于2025-02-10）。[4] Alessio Buscemi。使用 *ChatGPT 3.5* 对 *10* 种编程语言进行代码生成的比较研究。2023年。arXiv：2308.04477 [cs.SE]。网址：https://arxiv.org/abs/2308.04477 （访问于2025-02-10）。[5] Benjamin Canou、Grégoire Henry、Çagdas Bozman 和 Fabrice Le Fessant。“Learn OCaml：面向 OCaml 的在线学习中心”。载于：*2016*年 *OCaml* 用户与开发者研讨会。日本奈良，2016年9月。网址：https://inria.hal.science/hal-01352015（访问于2025-02-10）。[6] Federico Cassano、John Gouwar、Francesca Lucchetti、Claire Schlesinger、Anders Freeman、Carolyn Jane Anderson、Molly Q Feldman、Michael Greenberg、Abhinav Jangda 和 Arjun Guha。“将知识从高资源编程语言迁移至低资源编程语言，以改进代码大语言模型”。载于：*ACM* 编程语言论文集 8.OOPSLA2（2024年10月）。DOI：10.1145/3689735。[7] Federico Cassano、John Gouwar、Daniel Nguyen、Sydney Nguyen、Luna Phipps-Costin、Donald Pinckney、Ming-Ho Yee、Yangtian Zi、Carolyn Jane Anderson、Molly Q Feldman、Arjun Guha、Michael Greenberg 和 Abhinav Jangda。*MultiPL-E*：一种可扩展且易于扩展的神经代码生成基准测试方法。2022年。arXiv：2208.08227 [cs.LG]。网址：https://arxiv.org/abs/2208.08227（访问于2025-02-10）。[8] Alana Ceci、Hanneli C. A. Tavante、Brigitte Pientka 和 Xujie Si。“Learn-OCaml 编程平台的数据收集：对学生如何开发带类型的函数式程序进行建模”。载于：第*52*届 *ACM* 计算机科学教育技术研讨会论文集。SIGCSE ’21。线上活动，美国：计算机协会，2021年，第 1341 页。ISBN：978-1-4503-8062-1。DOI：10.1145/3408877.3439579。

[9] Mark Chen、Jerry Tworek、Heewoo Jun、Qiming Yuan、Henrique Ponde de Oliveira Pinto、Jared Kaplan、Harri Edwards、Yuri Burda、Nicholas Joseph、Greg Brockman、Alex Ray、Raul Puri、Gretchen Krueger、Michael Petrov、Heidy Khlaaf、Girish Sastry、Pamela Mishkin、Brooke Chan、Scott Gray、Nick Ryder、Mikhail Pavlov、Alethea Power、Lukasz Kaiser、Mohammad Bavarian、Clemens Winter、Philippe Tillet、Felipe Petroski Such、Dave Cummings、Matthias Plappert、Fotios Chantzis、Elizabeth Barnes、Ariel Herbert-Voss、William Hebgen Guss、Alex Nichol、Alex Paino、Nikolas Tezak、Jie Tang、Igor Babuschkin、Suchir Balaji、Shantanu Jain、William Saunders、ChristopherHesse、Andrew N. Carr、Jan Leike,

Josh Achiam、Vedant Misra、Evan Morikawa、Alec Radford、Matthew Knight、Miles Brundage、Mira Murati、Katie Mayer、Peter Welinder、Bob McGrew、Dario Amodei、Sam McCandlish、Ilya Sutskever 和 Wojciech Zaremba。《在代码上训练的大语言模型评估》。2021年。arXiv：2107.03374 [cs.LG]。网址：https://arxiv.org/abs/2107.03374 （访问于 2025年12月1日）。

[10] Yiu Wai Chow、Luca Di Grazia与Michael Pradel。“PyTy：修复Python中的静态类型错误”。载于：*IEEE/ACM* 第*46*届国际软件工程会议论文集。ICSE '24。葡萄牙里斯本：计算机协会，2024年。isbn：979-8-4007-0217-4。doi：10.1145/3597503.3639184。[11] Tristan Coignion、Clément Quinton与Romain Rouvoy。“大语言模型生成的代码在Leetcode上的性能研究”。载于：第*28*届软件工程评估与考核国际会议论文集。EASE '24。意大利萨莱诺：计算机协会，2024年，第79–89页。isbn：979-8-4007-1701-7。doi：10.1145/3661167.3661221。[12] Paul Denny、Viraj Kumar与Nasser Giacaman。“与Copilot对话：探索利用自然语言解决计算机科学入门课程问题的提示工程”。载于：第*54*届*ACM* 计算机科学教育技术研讨会论文集 第*1*卷。SIGCSE 2023。加拿大安大略省多伦多：计算机协会，2023年，第1136–1142页。isbn：978-1-4503-9431-4。doi：10.1145/3545945.3569823。[13] Xueying Du、Mingwei Liu、Kaixin Wang、Hanlin Wang、Junwei Liu、Yixuan Chen、Jiayi Feng、Chaofeng Sha、Xin Peng与Yiling Lou。“在类级代码生成中评估大语言模型”。载于：*IEEE/ACM* 第*46*届国际软件工程会议论文集。ICSE '24。葡萄牙里斯本：计算机协会，2024年。isbn：979-8-4007-0217-4。doi：10.1145/3597503.3639219。[14] Zhiyu Fan、Xiang Gao、Martin Mirchev、Abhik Roychoudhury与Shin Hwei Tan。“对来自大语言模型的程序进行自动修复”。载于：第*45*届国际软件工程会议论文集。ICSE '23。澳大利亚维多利亚州墨尔本：IEEE出版社，2023年，第1469–1481页。isbn：978-1-6654-5701-9。doi：10.1109/ICSE48619.2023.00128。[15] Laura Farinetti与Luca Cagliero。“对ChatGPT的批判性方法：SQL学习中的一次经验”。载于：第*56*届*ACM* 计算机科学教育技术研讨会论文集 第*1*卷。SIGCSETS 2025。美国宾夕法尼亚州匹兹堡：计算机协会，2025年，第318–324页。isbn：979-8-4007-0531-1。doi：10.1145/3641554.3701932。[16] Michael Fu、Chakkrit Tantithamthavorn、Trung Le、Van Nguyen与Dinh Phung。“VulRepair：基于T5的自动化软件漏洞修复”。载于：第*30*届*ACM*联合欧洲软件工程会议暨软件工程基础研讨会论文集。ESEC/FSE 2022。新加坡，新加坡：计算机协会，2022年，第935–947页。isbn：978-1-4503-9413-0。doi：10.1145/3540250.3549098。

[17] Jörg von Garrel与Jana Mayer。“学业中的人工智能——德国学生使用ChatGPT及人工智能工具的情况”。载于：人文与社会科学传播 10.1（2023年11月），第799页。issn：2662-9992。doi：10.1057/s41599-023-02304-7。[18] Sumit Gulwani、Ivan Radiček与Florian Zuleger。“面向入门编程作业的自动聚类与程序修复”。载于：第*39*届*ACM SIGPLAN*编程语言设计与实现会议论文集。PLDI 2018。美国宾夕法尼亚州费城：计算机协会，2018年，第465–480页。isbn：978-1-4503-5698-5。doi：10.1145/3192366.3192387。[19] Aliya Hameer与Brigitte Pientka。“使用自动评分教授函数式编程的艺术（经验报告）”。载于：*ACM* 编程语言汇刊3.ICFP（2019年7月）。doi：10.1145/3341719。[20] Arto Hellas、Juho Leinonen与Leo Leppänen。“将大语言模型聊天机器人融入课堂的经验”。载于：*2024*年*ACM*虚拟全球计算教育会议论文集 第*1*卷。SIGCSE Virtual 2024。美国北卡罗来纳州线上活动：计算机协会，2024年，第46–52页。isbn：979-8-4007-0598-4。doi：10.1145/3649165.3690101。[21] Yang Hu、Umair Z. Ahmed、Sergey Mechtaev、Ben Leong与Abhik Roychoudhury。“应用于编程作业的基于重构的程序修复”。载于：第*34*届*IEEE/ACM*自动化软件工程国际会议论文集。ASE ’19。加利福尼亚州圣迭戈：IEEE出版社，2020年，第388–398页。isbn：978-1-7281-2508-4。doi：10.1109/ASE.2019.00044。[22] Carlos E. Jimenez、John Yang、Alexander Wettig、Shunyu Yao、Kexin Pei、Ofir Press与Karthik Narasimhan。*SWE-bench*：语言模型能否解决真实世界的*GitHub*问题？ 2024年。arXiv：2310.06770 [cs.CL]。url：https://arxiv.org/abs/2310.06770（访问于2025-07-10）。[23] Harshit Joshi、José Cambronero Sanchez、Sumit Gulwani、Vu Le、Ivan Radiček与Gust Verbruggen。“修复近乎生成：使用大语言模型进行多语言程序修复”。载于：第三十七届*AAAI*人工智能会议暨第三十五届人工智能创新应用会议暨第十三届人工智能教育进展研讨会论文集。AAAI’23/IAAI’23/EAAI’23。AAAI出版社，2023年。isbn：978-1-5773-5880-0。doi：10.1609/aaai.v37i4.25642。[24] Samia Kabir、David N. Udo-Imeh、Bonan Kou与Tianyi Zhang。“Stack Overflow过时了吗？一项关于ChatGPT回答Stack Overflow问题之特征的实证研究”。载于：*2024*年*CHI*计算系统中人因会议论文集。CHI ’24。美国夏威夷州檀香山：计算机协会，2024年。isbn：979-8-4007-0330-0。doi：10.1145/3613904.3642596。

[25] Hieke Keuning、Isaac Alpizar-Chacon、Ioanna Lykourentzou、Lauren Beehler、Christian Köppe、Imke de Jong 和 Sergey Sosnovsky。“学生对不同计算机课程中生成式人工智能工具的看法及使用情况”。载于：第 *24* 届 *Koli Calling* 国际计算机教育研究会议论文集。Koli Calling ’24。美国纽约州纽约市：计算机协会，2024年。isbn：979-8-4007-1038-4。doi：10.1145/3699538.3699546。[26] Sophia D. Kolak、Ruben Martins、Claire Le Goues 和 Vincent Josua Hel- lendoorn。“语言模型的补丁生成：可行性与扩展行为”。载于：面向代码的深度学习研讨会。2022年。url：https://par.nsf.gov/biblio/10340618 （访问于 2025-02-10）。[27] Kaivaram Sivarama Krishna、Bommala Deepak Kumar、Morumpalli Deekshith Reddy、Bheema Sai Varun 和 Meena Belwal。“使用文本 Bison-001 学习编程：面向初学者的 Python、C、Java 解释器”。载于：*2024* 年第 *15* 届计算、通信与网络技术国际会议（*ICCCNT*）。2024年，第 1–6 页。doi：10.1109/ICCCNT61001.2024.10724566。[28] Juho Leinonen、Paul Denny、Stephen MacNeil、Sami Sarsa、Seth Bernstein、Joanne Kim、Andrew Tran 和 Arto Hellas。“比较学生与大语言模型生成的代码解释”。载于：*2023* 年计算机科学教育创新与技术会议论文集，第 *1* 卷。ITiCSE 2023。芬兰图尔库：计算机协会，2023年，第 124–130 页。isbn：979-8-4007-0138-2。doi：10.1145/3587102.3588785。[29] Arun-Balajiee Lekshmi-Narayanan、Priti Oli、Jeevan Chapagain、Mohammad Hassany、Rabin Banjade、Peter Brusilovsky 和 Vasile Rus。“在入门编程课程中解释代码示例：大语言模型与人类的比较”。载于：*2024 AAAI*人工智能会议论文集。由 Muktha Ananda、Debshila Basu Malick、Jill Burstein、Lydia T. Liu、Zitao Liu、James Sharpnack、Zichao Wang 和 Serena Wang 编辑。第 257 卷。神经信息处理系统论文集。PMLR，2024年2月26–27日，第 107–117 页。url：https://proceedings.mlr.press/v257/lekshmi-narayanan24a.html（访问于 2025-07-10）。[30] Fengjie Li、Jiajun Jiang、Jiajun Sun 和 Hongyu Zhang。“结合大语言模型与程序分析的混合自动化程序修复”。载于：*ACM* 软件工程与方法学汇刊 34.7（2025年8月）。issn：1049-331X。doi：10.1145/3715004。[31] Jiawei Liu、Chunqiu Steven Xia、Yuyao Wang 和 Lingming Zhang。“你的代码真的是 ChatGPT 生成的吗？对代码生成大语言模型的严格求值”。载于：第 *37* 届神经信息处理系统国际会议论文集。NIPS ’23。美国路易斯安那州新奥尔良：Curran Associates Inc.，2023年。url：https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf （访问于 2025-07-10）。

[32] Shuai Lu、Nan Duan、Hojae Han、Daya Guo、Seung-won Hwang 和 Alexey Svyatkovskiy。“ReACC：检索增强代码补全框架”。载于：计算语言学协会第 *60* 届年会论文集（第 *1* 卷：长篇论文），*ACL 2022*，爱尔兰都柏林，*2022* 年*5*月*22–27*日。由 Smaranda Muresan、Preslav Nakov 和 Aline Villavicencio 编辑。计算语言学协会，2022年，第 6227–6240 页。doi：10.18653/V1/2022.ACL-LONG.431。[33] Wenhan Lyu、Yimeng Wang、Tingting (Rachel) Chung、Yifan Sun 和 Yixuan Zhang。“评估大语言模型在计算机科学入门教育中的有效性：为期一学期的实地研究”。载于：第十一届 *ACM Learning @ Scale* 会议论文集。L@S '24。美国佐治亚州亚特兰大：计算机协会，2024年，第 63–74 页。isbn：979-8-4007-0633-2。doi：10.1145/3657604.3662036。[34] Stephen MacNeil、Andrew Tran、Arto Hellas、Joanne Kim、Sami Sarsa、Paul Denny、Seth Bernstein 和 Juho Leinonen。“在网络软件开发电子书中使用大语言模型生成的代码解释的经验”。载于：第 *54* 届 *ACM* 计算机科学教育技术研讨会论文集，第 *1* 卷。SIGCSE 2023。加拿大安大略省多伦多：计算机协会，2023年，第 931–937 页。isbn：978-1-4503-9431-4。doi：10.1145/3545945.3569785。[35] Anders Miltner、Adrian Trejo Nuñez、Ana Brendel、Swarat Chaudhuri 和 Isil Dillig。“使用天使式执行自底向上综合递归函数式程序”。载于：*ACM* 编程语言论文集6.POPL（2022年1月）。doi：10.1145/3498682。[36] Mihran Miroyan、Chancharik Mitra、Rishi Jain、Gireeja Ranade 和 Narges Norouzi。“利用助教反馈分析大语言模型响应对学生实时提问的教学质量与效率”。载于：第 *56* 届 *ACM* 计算机科学教育技术研讨会论文集，第 *1* 卷。SIGCSETS 2025。美国宾夕法尼亚州匹兹堡：计算机协会，2025年，第 770– 776 页。isbn：979-8-4007-0531-1。doi：10.1145/3641554.3701965。[37] Goda Nagakalyani、Saurav Chaudhary、Varsha Apte、Ganesh Ramakrishnan 和 Srikanth Tamilselvam。“面向入门编程作业的人工智能辅助评分工具的设计与求值：经验报告”。载于：第 *56* 届 *ACM* 计算机科学教育技术研讨会论文集，第 *1* 卷。SIGCSETS 2025。美国宾夕法尼亚州匹兹堡：计算机协会，2025年，第 805–811 页。isbn：979-8-4007-0531-1。doi：10.1145/3641554.3701913。[38] Noor Nashid、Mifta Sintaha 和 Ali Mesbah。“面向代码相关少样本学习的基于检索的提示选择”。载于：第 *45* 届国际软件工程会议论文集。ICSE '23。澳大利亚维多利亚州墨尔本：IEEE出版社，2023年，第 2450–2462 页。isbn：978-1-6654-5701-9。doi：10.1109/ICSE48619.2023.00205。

[39] Peter-Michael Osera 和 Steve Zdancewic。《类型与示例驱动的程序综合》。载于：PLDI '15。美国俄勒冈州波特兰：计算机协会，2015年，第619–630页。ISBN：978-1-4503-3468-6。DOI：10.1145/2737924.2738007。[40] Yun Peng、Shuzheng Gao、Cuiyun Gao、Yintong Huo 和 Michael Lyu。《领域知识至关重要：利用修复模板改进提示词，以修复 Python 类型错误》。载于：*IEEE/ACM 第46届软件工程国际会议论文集*。ICSE '24。葡萄牙里斯本：计算机协会，2024年。ISBN：979-8-4007-0217-4。DOI：10.1145/3597503.3608132。[41] Siddhartha Prasad、Ben Greenman、Tim Nelson 和 Shriram Krishnamurthi。《轻而易举地生成程序：学生对大语言模型的使用》。载于：*ACM 全球计算教育会议论文集，第1卷*。CompEd 2023。印度海得拉巴：计算机协会，2023年，第126–132页。ISBN：979-8-4007-0048-4。DOI：10.1145/3576882.3617921。[42] James Prather、Juho Leinonen、Natalie Kiesler、Jamie Gorson Benario、Sam Lau、Stephen MacNeil、Narges Norouzi、Simone Opel、Vee Pettit、Leo Porter、Brent N. Reeves、Jaromir Savelka、David H. Smith、Sven Strickroth 和 Daniel Zingaro。《超越炒作：生成式人工智能研究、教学实践与工具的当前趋势综述》。载于：*2024年计算机科学教育创新与技术工作组报告*。ITiCSE 2024。意大利米兰：计算机协会，2025年，第300–338页。ISBN：979-8-4007-1208-1。DOI：10.1145/3689187.3709614。[43] Julian Aron Prenner、Hlib Babii 和 Romain Robbes。《OpenAI 的 Codex 能修复错误吗？对 QuixBugs 的评估》。载于：第三届自动化程序修复国际研讨会论文集。APR '22。宾夕法尼亚州匹兹堡：计算机协会，2022年，第69–75页。ISBN：978-1-4503-9285-3。DOI：10.1145/3524459.3527351。[44] Francisco Ribeiro、José Nuno Castro de Macedo、Kanae Tsushima、Rui Abreu 和 João Saraiva。《由 GPT-3 驱动的类型错误调试：探究大语言模型在代码修复中的应用》。载于：第16届 *ACM SIGPLAN* 软件语言工程国际会议论文集。SLE 2023。葡萄牙卡斯凯什：计算机协会，2023年，第111–124页。ISBN：979-8-4007-0396-6。DOI：10.1145/3623476.3623522。[45] Georgios Sakkas、Madeline Endres、Benjamin Cosman、Westley Weimer 和 Ranjit Jhala。《通过解析式程序修复提供类型错误反馈》。载于：第41届 *ACM SIGPLAN* 编程语言设计与实现会议论文集。PLDI 2020。英国伦敦：计算机协会，2020年，第16–30页。ISBN：978-1-4503-7613-6。DOI：10.1145/3385412.3386005。[46] Sofia Santos、João Saraiva 和 Francisco Ribeiro。《利用大语言模型自动修复 Haskell 类型错误》。载于：第5届 *ACM/IEEE* 自动化程序修复国际研讨会论文集。APR '24。葡萄牙里斯本：计算机协会，2024年，第42–45页。ISBN：979-8-4007-0577-9。DOI：10.1145/3643788.3648012。

[47] Sami Sarsa、Paul Denny、Arto Hellas 和 Juho Leinonen。《使用大语言模型自动生成编程练习和代码解释》。载于：*2022*年 *ACM* 国际计算机教育研究会议论文集，第*1*卷。ICER '22。瑞士卢加诺和线上活动：计算机协会，2022年，第27–43页。ISBN：978-1-4503-9194-8。DOI：10.1145/3501385.3543957。[48] Dominik Sobania、Martin Briesch、Carol Hanna 和 JustynaPetke。《ChatGPT 自动错误修复性能分析》。载于：*2023 IEEE/ACM* 自动化程序修复国际研讨会（*APR*）。美国加利福尼亚州洛斯阿拉米托斯：IEEE计算机学会，2023年5月，第23–30页。DOI：10.1109/APR59189.2023.00012。[49] Nigar Surameery 和 Mohammed Shakor。《使用 Chat GPT 解决编程错误》。载于：《国际信息技术与计算机工程期刊》（2023年1月），第17–22页。DOI：10.55529/ijitc.31.17.22。[50] Priyan Vaithilingam、Tianyi Zhang 和 Elena L. Glassman。《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：*2022*年 *CHI*计算机系统人因会议扩展摘要集。CHI EA '22。美国路易斯安那州新奥尔良：计算机协会，2022年。ISBN：978-1-4503-9156-6。DOI：10.1145/3491101.3519665。[51] Tung Do Viet 和 Konstantin Markov。《使用大语言模型进行错误定位与修复》。载于：*2023*年第*12*届国际意识科学与技术会议（*iCAST*）。2023年，第192–197页。DOI：10.1109/iCAST57874.2023.10359304。[52] Chong Wang、Jian Zhang、Yebo Feng、Tianlin Li、Weisong Sun、Yang Liu 和 Xin Peng。《让代码大语言模型在代码库级代码生成中使用自动补全工具》。载于：*ACM*软件工程与方法论汇刊 34.7（2025年8月）。ISSN：1049-331X。DOI：10.1145/3714462。[53] Weishi Wang、Yue Wang、Shafiq Joty 和 Steven C.H. Hoi。《RAP-Gen：结合 CodeT5 的检索增强补丁生成，用于自动程序修复》。载于：第*31*届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集。ESEC/FSE 2023。美国加利福尼亚州旧金山：计算机协会，2023年，第146–158页。ISBN：979-8-4007-0327-0。DOI：10.1145/3611643.3616256。[54] Michel Wermelinger。《使用 GitHub Copilot 解决简单编程问题》。载于：第*54*届 *ACM* 计算机科学教育技术研讨会论文集，第*1*卷。SIGCSE 2023。加拿大安大略省多伦多：计算机协会，2023年，第172–178页。ISBN：978-1-4503-9431-4。DOI：10.1145/3545945.3569830。[55] An Yang、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chengyuan Li、Dayiheng Liu、Fei Huang、Haoran Wei、Huan Lin、Jian Yang、Jianhong Tu、Jianwei Zhang、Jianxin Yang、Jiaxi Yang、Jingren Zhou、Junyang Lin、Kai Dang、Keming Lu、Keqin Bao、Kexin Yang、Le Yu、Mei Li、Mingfeng Xue、

Pei Zhang、Qin Zhu、Rui Men、Runji Lin、Tianhao Li、Tianyi Tang、Tingyu Xia、Xingzhang Ren、Xuancheng Ren、Yang Fan、Yang Su、Yichang Zhang、Yu Wan、Yuqiong Liu、ZeyuCui、Zhenru Zhang 和 Zihan Qiu。*Qwen2.5* 技术报告。2025年。arXiv：2412.15115 [cs.CL]。网址：https://arxiv.org/abs/2412.15115（访问日期：2025-02-10）。

[56] Boyang Yang、Haoye Tian、Weiguo Pian、Haoran Yu、Haitao Wang、JacquesKlein、Tegawendé F. Bissyandé 和 Shunfu Jin。《CREF：面向编程辅导工具的基于大语言模型的对话式软件修复框架》。载于：第*33*届 *ACM SIGSOFT* 国际软件测试与分析研讨会论文集。ISSTA 2024。奥地利维也纳：计算机协会，2024年，第882–894页。ISBN：979-8-4007-0612-7。DOI：10.1145/3650212.3680328。[57] Stephanie Yang、Hanzhang Zhao、Yudian Xu、Karen Brennan 和 Bertrand Schneider。《借助人工智能辅导工具进行调试：探究新手求助行为与感知学习》。载于：*2024*年 *ACM* 国际计算教育研究会议论文集——第*1*卷。ICER '24。澳大利亚维多利亚州墨尔本：计算机协会，2024年，第84–94页。ISBN：979-8-4007-0475-8。DOI：10.1145/3632620.3671092。[58] Burak Yetistiren、Isik Ozsoy 和 Eray Tuzun。《评估 GitHub Copilot 代码生成的质量》。载于：第*18*届国际软件工程预测模型与数据分析会议论文集。PROMISE 2022。新加坡：计算机协会，2022年，第62–71页。ISBN：978-1-4503-9860-2。DOI：10.1145/3558489.3559072。[59] Wei Yuan、Quanjun Zhang、Tieke He、Chunrong Fang、Nguyen Quoc Viet Hung、Xiaodong Hao 和 Hongzhi Yin。《CIRCLE：跨编程语言的持续修复》。载于：第*31*届 *ACM SIGSOFT* 国际软件测试与分析研讨会论文集。ISSTA 2022。线上，韩国：计算机协会，2022年，第678–690页。ISBN：978-1-4503-9379-9。DOI：10.1145/3533767.3534219。[60] Yihan Zhang、Brigitte Pientka 和 Xujie Si。《资料：在函数式编程课程背景下评估大语言模型：一项综合研究》。Zenodo。2026年。DOI：10.5281/zenodo.18470483。[61] Qinkai Zheng、Xiao Xia、Xu Zou、Yuxiao Dong、Shan Wang、Yufei Xue、Lei Shen、Zihan Wang、Andi Wang、Yang Li、Teng Su、Zhilin Yang 和 Jie Tang。《CodeGeeX：一种用于代码生成的预训练模型及其在 HumanEval-X 上的多语言基准测试》。载于：第*29*届 *ACM SIGKDD* 知识发现与数据挖掘会议论文集。KDD '23。美国加利福尼亚州长滩：计算机协会，2023年，第5673–5684页。ISBN：979-8-4007-0103-0。DOI：10.1145/3580305.3599790。

关于作者

Yihan Zhang 是加拿大麦吉尔大学的一名硕士生。请通过以下方式联系她：yihan.zhang2@mail.mcgill.ca。
https://orcid.org/0009-0002-9107-6774

Brigitte Pientka 是加拿大麦吉尔大学的教授。请通过以下邮箱联系她：brigitte.pientka@mcgill.ca。
https://orcid.org/0000-0002-2549-4276

Xujie Si是加拿大多伦多大学的教授。请通过以下邮箱联系他：six@cs.toronto.edu。
https://orcid.org/0000-0002-3739-2269