

代码辅助的 LLMs 在数学上评估基于事实的推理

Zena Al-Khalili

Nick Howell

Dietrich Klakow

Saarland Informatics Campus, Saarland University, Germany

{ zakhalili, nhowell, dietrich.klakow } @lsv.uni-saarland.de

Abstract

辅助大型语言模型进行代码生成提高了它们在数学推理任务上的表现。然而，对代码辅助大型语言模型的评估通常仅限于执行正确性，缺乏对其生成程序的严格评估。在这项工作中，我们通过对代码辅助大型语言模型生成的程序响应数学推理任务进行深入分析来缩小这一差距。我们的评估重点在于大型语言模型在多大程度上将其程序依据数学规则，以及这如何影响其最终表现。为此，我们对五种不同的大型语言模型在两个不同的数学数据集中的生成进行手动和自动评估。我们的结果揭示了根据模型能力和数学问题难度大型语言模型的程序依据分布。此外，数学依据对闭源模型更有效，而开源模型未能正确地在其解决方案中使用数学规则。在 MATH500 数据集中，依据的程序百分比减少到一半，而与 ASDiv 小学问题相比，非依据生成程序增加了一倍。我们的工作强调了需要在执行准确性指标之外进行深入评估，以更好地理解代码辅助大型语言模型在数学领域的能力和限制。

最近，大型语言模型 (LLMs) 在诸如数学推理等复杂推理任务上取得了卓越的表现，这得益于其规模和多步骤推理方法，特别是链式推理 (CoT) (?)，该方法要求 LLM 在生成最终答案之前生成显式的推理步骤。

尽管其取得了成功，但对 CoT 推理步骤的研究揭示了 LLMs 的关键缺陷，如犯计算错误 (?) 和生成误报链 (?)，即：包含推理错误但生成正确的最终答案。

以代码辅助推理的方法 (??????)，通过指导大型语言模型 (LLMs) 生成程序化推理步骤 (例如 Python 程序)，并将这些程序的执行交由解释器来解决这些问题，这样可以避免计算错误并确保真实性。此类方法已被发现进一步提高了 LLMs 在数学推理任务上的表现。

然而，性能提升主要是通过执行结果的正确性来评估的，而不是通过生成程序的质量及其对推理过程的影响来评估。这是有问题的，因为生成的程序可能依赖于穷举搜索或记忆的

信息来生成正确的答案，而不是将推理过程建立在领域概念上，例如数学规则，从而生成不易推广且难以验证和信任的程序。图 1 显示了 GPT-4 和 StarCoder2 在解决 ASDiv 数据的数学问题时生成的此类程序。

本研究的目标是通过分析代码辅助的大型语言模型在解决数学任务时生成的中间程序，以评估其实际的推理能力。

在此评估中，我们主要关注代码辅助的 LLM 在生成解决方案时，将其推理过程基于领域概念和数学规则的程度，以及这种基础 (或其缺失) 如何影响其在数学任务中的最终表现。

代码通过库 (例如: Sympy) 提供对如数学规则等领域概念的访问，这可以帮助 LLM 生成根植于世界知识 (即: 数学规则) 的程序，更接近于人类般的推理。鉴于此，我们期待根植性的程序值得信赖，更容易验证，并且可能具有更高的解决率，而非根植性的程序则更容易出错，难以信任，并降低最终性能。

我们通过寻找以下研究问题的答案来调查这在多大程度上是正确的。

- 研究问题 1: 代码辅助的 LLM 在多大程度上将其推理与相关的数学概念联系起来?
- RQ2: 代码辅助的 LLMs 如何实现这些概念?
- RQ3: 将数学概念应用于基础和数学任务的最终表现之间有什么关系?
- RQ4: 这些问题的答案在更具挑战性的数学问题中有多大变化?

为了回答这些问题，我们首先手动分析了由 GPT4o-mini、GPT4、Qwen2.5、Llama3 和 StarCoder2 生成的中间程序的一个子集，用于解决 ASDiv 和 MATH500 数据集中的数学问题，从而了解其基础和实现风格 (??)。根据手动分析的观察结果，我们设计了一种分类法，反映被评价模型所使用的不同基础层次 (??)。为了将分析扩展到生成程序的完整集合，我们采用了两种自动化评估方法：使用最新的 o3-mini

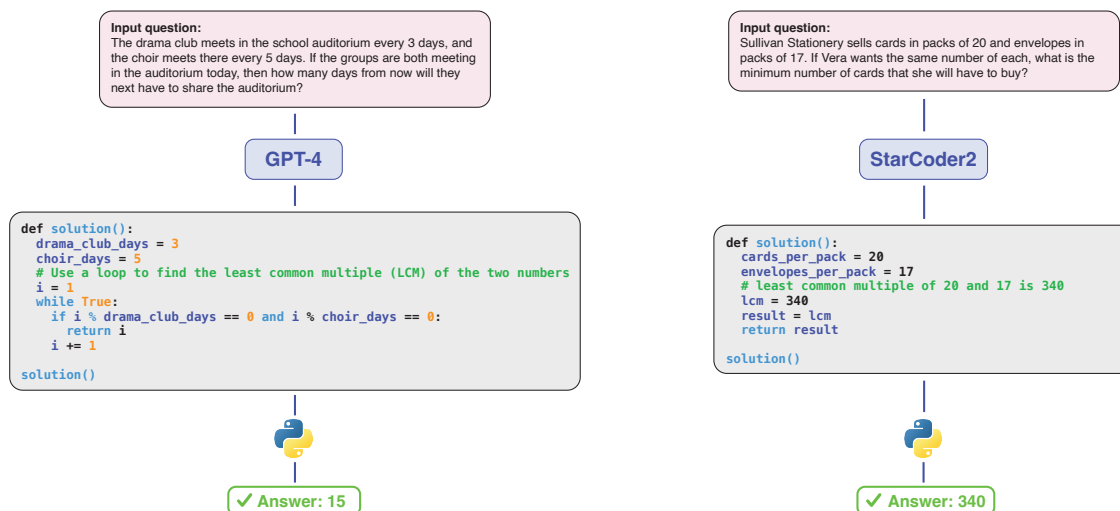


Figure 1: 由 GPT-4 生成的程序（左）使用穷举搜索找出输入问题的答案，而 StarCoder2（右）则依赖于记忆的信息，而不是生成解决输入问题的步骤。

模型的 LLM 法官，以及我们提出的方法，代码结构法官，该方法使用从程序的抽象语法树中提取的特征来训练决策树分类器 (??)。我们发现，代码结构法官达成与人类判断更高的一致性，比 LLM 法官更高，分别为 81% 和 73% (4.1)；因此，我们将其用于所有生成程序的大规模评估。

据我们所知，这是第一次分析在数学推理任务中代码辅助 LLMs 生成的程序。我们从第 (4.2) 节总结我们的收获如下：

1. 代码辅助的 LLM 通过调用数学库中的概念或从头实现这些概念，将其程序基于数学知识。
2. 代码辅助的大型语言模型也依赖于暴力搜索或记忆的信息来解决某些数学问题。
3. 基于数学知识的生成对 GPT 模型和 Qwen 来说更有效，而开源模型未能正确将其推理与有用的概念结合。
4. 在具有挑战性的 MATH500 数据上，基础分布发生了完全改变，迫使 GPT 模型和 Qwen 加倍依赖于暴力搜索和记忆的信息，代价是更具数学基础的解决方案减少了。

我们深入的评估强调了对 LLMs 生成的程序进行更全面评估的必要性，不仅仅是实施准确性指标，因为实施准确性指标并不能提供关于模型在数学领域实际能力和限制的有益结论。

1 相关工作

程序作为数学推理的中间步骤。诸如 (?????) 之类的代码辅助推理方法，提示 LLM 生成程

序而不是自然语言中的中间步骤。已经发现，这些方法提高了 LLM 在数学推理任务上的表现。除了情境学习方法之外，其他工作如 (??) 微调了中等规模的语言模型，如在代码推理路径上的 Code-Llama (?)，并在数学推理任务上取得了与闭源模型相当的表现。然而，无论是情境学习还是微调方法，主要都是通过最终答案的正确性进行评估，忽视了中间程序及其质量可能如何影响推理过程。本研究专注于探索这些程序，以准确评估代码辅助 LLM 的问题解决能力。

中间步骤的评估这项工作还涉及到一些研究，这些研究评估了自然语言 (????) 中 LLM 的中间推理步骤，针对多维评估维度，如鲁棒性和可信度，或在若干推理任务中识别和纠正错误。这些研究要么使用人写的参考链与被评估链进行比较，要么使用一个有能力的 LLM 来定位错误并判断推理链的质量。代码的中间步骤尚未被探索，因此我们旨在分析这些步骤的质量以及它们如何影响 LLM 的最终表现。

LLM 生成代码的评估。为了评估 LLM 生成的代码，先前的工作集中在基于测试的评估和功能正确性，例如 (??)，其他人 (???) 提出了度量，用来衡量生成结果与参考的人类编写代码的相似度，该结果获取成本较高，并且通常不考虑生成的多样性。(?) 使用一个功能强大的 LLM 作为审判者，在代码生成任务中定位生成程序中的错误。我们借鉴他们的方法，用 LLM-Judge 来评估数学推理任务中的生成程序。最后，(?) 分析了 LLM 生成的程序在代码生成任务中的特性，包括代码复杂性、API 调用数量和错误类型，即：导致程序失败的 bug。尽管见解丰富，该分析以一种逻辑错误

为结论，这是不同 LLM 共同分享的常见错误类型。在本文中，我们深入研究以了解 LLM 在编写解决数学问题代码时犯下的逻辑错误。

2 中间程序的评估

本次评估的主要重点是不仅限于端到端评估和单一分数指标，进一步评估代码辅助大型语言模型生成的程序。我们更进一步理解大型语言模型用于解决数学问题的逻辑。因此，我们评估它们在生成程序中使用数学概念的情况。数学概念是指用于解决特定数学问题的原理，例如：使用欧几里得算法来找到两个数的最大公因数。评估生成程序的数学概念使用情况类似于因为正确的原因而正确，因为在解决方案中使用数学概念能确保推理更符合人类的推理方式，并且更容易验证和信任。

2.1 评估设置

给定一个自然语言的数学问题，我们提示大型语言模型生成一个解决问题的 Python 程序。我们将删除那些在解析¹时失败的程序。在附录 D.1 中，我们提供了生成程序最常见错误的列表。

2.2 评估数据集

我们使用了来自两个数据集的数学问题，即 ASDiv (?) 和 MATH500 (??)。这两组数据包含了从小学到高中竞赛水平的各种问题。我们排除了 ASDiv 数据中的简单问题，只专注于更复杂的技能，比如解方程组、找出最大公约数或最小公倍数。另一方面，据报道 MATH500 集合对许多大语言模型 (??) 来说更具挑战性。因此，我们用它来研究大语言模型如何修改其生成的程序以解决更具挑战性的数学问题。

由于缺乏对大语言模型生成的数学推理任务代码的细粒度评估，我们进行了详细的人为手工分析，以回答我们前两个研究问题：大语言模型是否将其生成的程序与数学概念相结合，以及这种结合在生成的输出中是如何实现的。此分析仅使用了一部分评估数据集，其中包含从整个数据集中随机抽取的 300 个程序，包含两个主要步骤：

程序分区 我们将每个程序分为三个部分：(1) 转录：程序的这一部分将问题中的信息转录为变量。(2) 处理：在这一部分中，通过操作和函数调用对变量进行操作以得出答案。(3) 结果收集：返回处理部分的最终答案。有些部分可以合并为一行代码，例如在同一步中同时进行处理和返回结果。

¹我们解决了导入错误和全局缩进错误（整个主体缩进错误）。详见附录 A.2。

我们通过归纳法分析代码行的处理，考虑以下几个方面：逻辑，实现风格，一致性，验证工作量和可信度。

我们检查代码行是否被组织成一个连贯的步骤序列，这些步骤代表某个数学概念的有效实现，人类可以根据现有的可能实现来验证该概念。

另外，我们检查一个解决方案的所有步骤是否是明确生成的，而不是由模型推断的。推断步骤可能涉及模型无法正确计算的计算，因为模型在算术运算中存在 (???) 的已知缺陷，这使得验证和信任这些解决方案变得更加困难。

最重要的是，我们检查数学概念在生成的程序中是如何实现的。基础实现可以通过多种风格进行，或者仅仅通过基本运算来处理非常简单的数学问题和从头开始的实现，或者通过调用相关数学库中的函数，这些库函数代表了一种更抽象的解决方案形式。我们还观察到一些实现依赖于穷举循环，逐一搜索所有可能的答案，或者完全没有处理部分。

我们将所有这些观察到的趋势归纳成六个类别，这些类别构成了我们提出的与数学概念及其风格相关的分类体系。

根据手动分析的观测，我们建议将大语言模型 (LLM) 生成的程序分为六个互斥的类别，其中三个类别代表的是基于数学概念的程序，以不同的实现风格表现；而另外三个类别则代表非基于概念的程序，这些程序主要依靠记忆的信息或穷举搜索来找到最终答案。

1. 通过库调用的概念。通过调用相关的数学库、标准库或外部库来参考一个数学概念。
2. 原始程序：由于问题的简单性，解决方案仅用原始操作来表达，其中不能调用或实现任何库功能。
3. 从头实现库功能。模型并没有调用库函数，而是从头实现了相同的功能。该模型要么将此实现内联到生成的代码中，要么将其编写为一个自定义函数，以便在需要时调用。
4. 蛮力法：程序遍历所有可能的值以找到答案，而在搜索过程中不使用任何数学知识指导。
5. 杂乱无章：程序由无关的步骤组成，这些步骤似乎是前面课程的混合体。通常包括定义但未使用的变量，或相反的情况。
6. 无逻辑：这些程序完全跳过处理部分，仅仅返回结果而没有明确生成达到结果的步

骤。(将逻辑以自然语言评论的形式生成也被认为是无逻辑。)

扩展人类手动分析到整个评估数据集需要耗费大量的时间和精力。因此，我们通过训练分类器来自动化评估，这些分类器使用我们提出的分类法中的一个类别来标记生成的程序。为此，我们首先使用分类法中的类别标注一个训练集，然后我们使用从程序中提取的特征训练一个决策树分类器。我们将训练好的分类器与一个 LLM-Judge 进行比较，并在一个保留的已标注集上评估这两个判断器。我们选择更为准确的判断器进行大规模评估。

为了训练自动评估方法，两位作者使用分类法的标签对随机抽取的生成程序子集进行注释。标注的集合来自 ASDiv 分组的 300 个程序。210 个实例被用于训练，而其余用于测量自动评审的性能。标注指南是基于人工分析的观察结果开发的。在尝试不同的标注方案后，我们发现人类对每个程序标签的分配产生了低的标注者间一致性 (IAA)，而每行标注产生了非常高的一致性。因此，我们选择了每行标注方案，并采用一个简单的算法来根据每行标签分配程序级标签。关于标注指南和标注者背景的信息可以在附录 B 中找到。对 300 个样本中的 50 个进行了双重独立标注，并有 93% 行粒度和 100% 样本粒度的一致性。然后对争议进行裁定，并更新指南以解决不明确之处。

2.2.1 自动化评估方法

代码结构判断：决策树模型 我们注意到，分类法的每个类别往往依赖于代码结构的特定特征。因此，我们建议使用代码结构中的特征来训练一个决策树分类器，该分类器将用于评估生成的程序。训练分类器的特征是从每个程序的抽象语法树 (AST) 中提取的，这些特征包括：

- 函数调用次数，包括对模型自身写入的函数的调用次数。
- 导入语句的数量。
- 内置操作的数量。例如：+，<，...
- 控制流语句的数量：例如，if，for，break，...
- 定义但未使用的变量数量。使用但未定义的变量数量。

用于决策树模型的最大深度是 5。我们尝试了其他模型，如 SVM 和随机森林，但没有发现进一步的提升。另一方面，神经分类器则需

要更多的训练数据，因此需要更多的标注和人工努力。

继其他相关工作使用 LLM 作为裁判，我们采用 LLM 来评估给定的程序并为该程序分配一个分类类别。为此，我们在提示中提供了分类类别的详细描述，并要求 LLM-judge 仔细分析输入程序，最后要求其提供一个最适合该程序的类别编号。提示中不包含任何分类类别的参考程序。完整提示见附录。

我们试验了来自 OpenAI 中最先进的两个模型，即 GPT4o 和 O3-mini，发现 O3-mini 作为评估者取得了略好的结果。我们将模型的推理努力设定为高，并允许最多 20,000 个输出令牌，其中包括推理令牌。

3 实验装置

3.1 评估模型

我们分析了几个大型语言模型 (LLMs) 的代码生成，包括开源模型，如：StarCoder2 15B (?)、Llama 3.1 8B (?) 和 Qwen2.5 7B (?)。我们使用了这些模型的指令调优版本。此外，我们评估了来自 OpenAI 的 GPT-4 (?) 和 GPT4o-mini。所有模型均使用贪婪解码进行评估。实现细节在附录 A.1 中。

我们采用来自 (?) 的提示在 ASDiv 数据集上评估 LLMs，仅使用三个示例，而不是八个，因为使用完整的一组示例没有观察到进一步的收益。对于 MATH500 数据，我们使用来自数据集训练集的示例对 LLMs 进行提示，这些示例改编自 (?)。我们在零样本设置中评估 GPT4o-mini 和 Qwen，因为我们观察到使用这种设置模型倾向于生成更少的自然语言生成。在附录 A.3 中可以找到两个完整的提示。

4 结果与分析

4.1 不同自动化方法的比较

为了比较自动化评估方法，我们通过使用注释者为每个生成的程序分配的准确性来衡量与人工判断的一致性。

Code-Structure Judge 在四个不同的种子上取得了平均准确率 81%，标准差为 0.005。图 ?? (左) 展示了 ASDiv 数据中保留集上 Code-Structure Judge 的混淆矩阵。

我们注意到，Code-Structure 判断器在大多数类别上都达到了较高的精确度和召回率，除了 Disorganized 类别的召回率较低，为 0.16。Disorganized 程序可能是其他类别的混合体，这些类别之间的区别可能是语义上的，而不是结构上的。Primitives 类别的精确度也低于其他类别，为 0.72。附录 ?? 包括了一些被 Code-

Model	Conceptual	Primitive	FS Implementation	Brute-Force	Disorganized	No Logic
GPT4o-mini	90.5%	96.5%	86.3%	77.7%	93.7%	85.7%
GPT4	86%	86%	100%	68.4%	60%	63.6%
Qwen2.5	91.5%	78.3%	89.4%	100%	60%	100%
StarCoder2	48%	44.4%	76.4%	47.6%	30.7%	61.5%
Llama3.1	35.7%	48.1%	37.5%	53.7%	33.3%	57.5%

Table 1: 执行（宏观）准确性，以百分比表示所有评估模型在 ASDiv 数学问题上的每个分类的准确度。"FS" 代表从头开始。

Structure 判断器错误分类的程序，以及精确度、召回率和 F1-分数的完整表格。

而另一方面，LLM-judge 的准确率较低，仅为 73%。Primitives 类的精度为 0.56，而 Disorganized 和 No Logic 类的召回率分别为 0.25 和 0.37，明显较低，这使得该评判相比于 Code-Structure 评判的总体准确率降低了 8%。图 ?? (右) 显示了 LLM-judge 和真实标签的混淆矩阵对比。

我们对一些被错误分类为无逻辑类的实例进行了定性错误分析，发现 LLM-judge 倾向于将从问题中转录信息的步骤误认为是逻辑的一部分，从而将实例错误地分类为基础类。

4.2 从评估中间程序中获得的关于代码辅助 LLM 的见解

我们使用代码结构评估器对所有生成的数学文字题程序进行大规模评估，旨在回答我们的研究问题：

在 ASDiv 上的每个模型的分类型分布如图 2 所示。我们观察到，虽然所有模型都生成以数学为基础的程序，但它们在落实的方式上存在差异：GPT 模型和 Qwen 主要通过库调用来落实，而其他开源模型则通过原语来落实。所有模型较少依赖从头开始的实现。相比新模型（GPT4o-mini 14% 和 Qwen 7%），旧模型（StarCoder2, Llama3.1, 和 GPT4）生成更多没有基础的程序（分别为 29%，43%，和 26%）。GPT4 和 Llama3.1 大多数没有基础的生成是“暴力搜索”（分别占有所有生成程序的 20% 和 26%），而 StarCoder2 则产生更多“无逻辑”解决方案（占有所有生成的 16%）。所有模型生成的无序程序的百分比都很小。

将生成过程与数学规则联系起来与最终性能之间是什么关系？ 表格 1 提供了在 ASDiv 数学问题上评估的所有模型每个分类的执行宏观准确率。我们注意到，GPT 模型和 Qwen 的落地程序相比能力较弱的开源模型生成的落地程序显示出更高的准确率，这表明对这些模型来说，落地效果不是那么理想。另一方面，许多模型在非落地程序上表现良好；Llama3.1 和

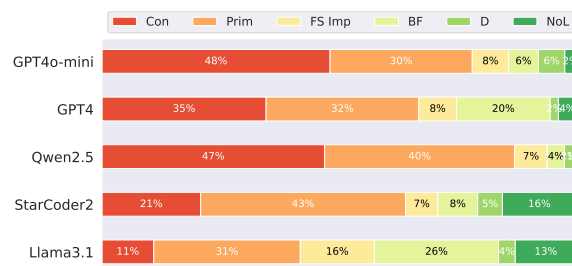


Figure 2: 在 ASDiv 问题上，各模型的分类型分布以百分比表示。类别有：'Con'：概念性的，'Prim'：基本的，'FS Imp'：从零开始实现，'BF'：暴力破解，'D'：无序的，以及'NoL'：无逻辑。

StarCoder2 在无逻辑的程序上的准确率高于概念性的程序。这意味着这些模型试图通过记住答案来解决数学问题，而不是解决其背后的推理问题。同样，许多模型通过暴力搜索来找到最终答案，而没有通过任何数学规则引导搜索。研究执行准确率与落地之间的相关性没有提供有用的见解，因为模型似乎在所有类别上表现良好。² 总体来看，我们可以看出对于 GPT 模型和 Qwen 来说，将生成的内容落地到数学知识上更为有效。相反，开源模型依赖于其他技巧来解决数学问题，这可能表明它们无法进行成功的落地。

为什么 LLMs 在同一类中表现不同？ GPT4o-mini 在混乱类上的精确度非常高，这与所有其他模型不同。为了进一步理解这一反直觉的现象，我们分析了 GPT4o-mini 在这一类中的 16 生成结果，发现所有生成结果中，14 是误判，其中包括 9 原始和 5 无逻辑。此外，我们还调查了 Llama3.1 和 StarCoder2 为什么在基础类上通常表现不佳。在概念类上，我们对两个模型的 20 个程序进行了质量分析，发现有 18/20 个程序调用了错误的概念，这表明调用相关数学概念的能力较差。其他 2/20 个程序从问题中错误记录信息。此外，在原始类上，我们对由两个模型生成的 20 个程序进行了错误分析，发现 9/20 是混乱和无逻辑的误

²由于 GPT 模型和 Qwen 未校准生成的样本量较少，我们无法对其在这些类别中的表现得出太多结论。

Model	Conceptual	Primitive	FS Implementation	Brute-Force	Disorganized	No Logic
GPT4o-mini	56.5%	59.2%	48.7%	48.6%	64.2%	56.8%
Qwen2.5	48.6%	54.2%	54.5%	60.2%	23.5%	62.5%
StarCoder2	10%	17.1%	0.0%	25.0%	8.3%	46.1%

Table 2: 执行（宏）精度，按分类法类别以百分比表示，适用于在 MATH500 问题上评估的三个模型。“FS”表示从头开始。

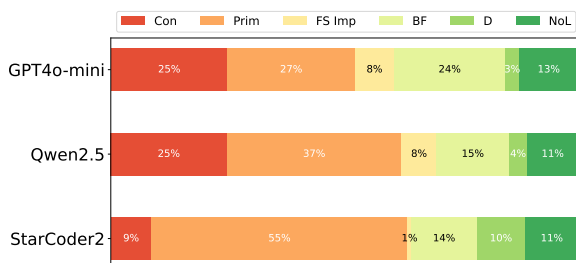


Figure 3: MATH500 数据集中所有评估模型的分类型分布百分比。类别包括：‘Con’：概念，‘Prim’：原始，‘FS Imp’：从头实现，‘BF’：暴力，‘D’：无组织和‘NoL’：无逻辑。

判，8/20 有推断步骤错误，模型推断一个步骤的计算而不是生成计算步骤，导致错误在代码中的其余部分传播。此外，2/20 从问题中错误记录信息，1/20 的正确答案错误。最后，在从零开始的实现类中，Llama3.1 相比其他模型具有最低精确度。检查来自这一类的 10 个随机生成结果，我们发现模型实现了不同于问题需要的解决方案。例如，当问题要求的是最小公倍数时，该问题实现了最大公约数代码，反之亦然。另一方面，StarCoder2 则成功实现了许多解决方案。

更具挑战性的数学问题如何影响 LLM 的基础？ 为了回答这个问题，我们评估了 GPT4o-mini、Qwen2.5 和 StarCoder2 对 MATH500 数据集中的问题生成的程序，该数据集被发现对于大型语言模型来说更具挑战性(?)。图 3 展示了每个在 MATH500 上评估的 LLM 的分类型分布。

我们注意到，更具挑战性的数据改变了基础生成的分布。所有模型的概念程序数量减少了一半，而所有模型中的非基础程序，尤其是穷举法和无逻辑程序，数量翻了一番。这表明问题的难度可能导致模型减少对基础程序的依赖。

随着无逻辑类程序数量的增加，我们对这个类的一些生成项进行定性分析，发现对于 StarCoder2 和 Qwen2.5，许多实例实际上直接返回答案，而对于 GPT-4o-mini，许多实例是通过自然语言中的注释而不是可以执行以获得答案的程序步骤来解决问题的。我们仍然认为

这些实例属于无逻辑类，因为它们类似于自然语言中的推理，而代码在寻找正确答案上没有有效的帮助。文本推理优于代码这一偏好可能是由于问题的复杂性，正如在(?)中经过实证研究和讨论的那样，他们展示了一些 GPT 模型根据任务的复杂性更喜欢文本推理而不是代码推理。

关于执行准确性与基础的关系，表 2 展示了在 MATH500 问题上每个模型每类问题的执行宏准确率。我们观察到，非基础化程序的准确率要高于或与基础化生成的准确率相当，这在两个开源模型³中表现得更为明显。这是一个问题，因为依赖于记忆的信息或蛮力搜索来找到答案是不可靠的，并且无法提供一致的性能。我们还研究了由评估的 LLMs 在两个数据集上所进行的库调用，发现 GPT4o-mini 和 Qwen 在两个数据集中利用了更多种类的库调用，但是这些库调用的分布从 ASDiv (math.gcd、abs、math.lcm、min 是最常见的) 转向 MATH500 (sympy.solve 为主导)。另一方面，StarCoder2 并没有使用很多库，并且几乎没有在 MATH500 数据集上生成概念性程序。

最后，我们查看数学科目与分类类别之间的关系，以确认模型是否会根据给定问题的科目生成特定类型的程序。GPT4o-mini 和 Qwen2.5 倾向于使用更多原始解决方案来解决较容易的问题，例如预代数和预微积分。此外，它们对代数和概率问题采用概念化程序。值得注意的是，这两个模型还利用记忆信息或暴力搜索来解决中级代数、数论和一些代数问题。附录中的图 7 展示了 MATH500 数据集中每个数学科目的分类类别分布。另一方面，StarCoder2 在大多数科目中严重依赖原始解决方案，表明其在解决不同类型的数学问题时逻辑缺乏多样性。

在这项工作中，我们对代码辅助的 LLM 生成程序在数学问题上的表现进行了深入分析。我们的评估集中于 LLM 在生成程序中应用数学概念的程度，以及这如何影响它们的最终性能。我们将生成的程序分为六个不同的类别，这构成了我们提出的基于数学概念的分类法。

³GPT4o-mini 和 Qwen 在 Disorganized 类别上表现出高准确率是由于误报导致的。

我们使用分类法中的类别对程序语料库进行了注释，并训练了一个我们称之为代码结构评判的决策树模型。代码结构评判在评估中表现优于 LLM 评判基线，并被用于对更多生成程序的大规模评估。我们的研究表明，模型使用了扎实推理和未经扎实推理的推理，包括使用蛮力搜索和记忆化信息。未经扎实推理的方法的使用高度依赖于 LLM 的能力和问题的难度。我们的工作强调对代码辅助的 LLM 的整体评估是必要的，我们希望我们的发现能够激励未来的工作，进一步研究为什么 LLM 使用未经扎实推理的解决方案以及如何减轻这一现象。

我们注意到我们的工作存在局限性。首先：代码结构判断在无序类别上仍然不准确，导致许多误报。其次，我们依赖一种提示技术，没有比较使用提示来改进生成程序并进一步优化时的性能，例如自我优化 (?) 或基于代码的自我验证 (?)。

作者要感谢 Vagrant Gautam 的指导和有用的反馈。我们还感谢 Marius Mosbach 和 Ellie Pavlick 在这项工作的早期阶段提供的反馈。我们还感谢 Miaoran Zhang 的有益讨论和 Tural Mammadov 在实验基础设施上的帮助。作者得到了 DFG (德国研究基金会) 项目 232722074、SFB 1102 的资助。

A 实验细节

A.1 实现细节

我们使用来自 HuggingFace (?) 的 `starcoder2-15b-instruct`，`Meta-Llama-3-8B-Instruct`，`Qwen2.5-7B-Instruct`。对于 OpenAI 模型，我们使用 `gpt-4-32K` 和 `gpt-4o-mini`。我们对所有模型使用 `int8bit` 模型量化，但 OpenAI 模型除外，因为我们观察到模型执行精度没有显著差异。最终，我们使用 NVIDIA A100 40GB GPU，在 ASDiv 上以批量大小 = 32 评估 Llama3、StarCoder2 和 Qwen2.5，这大约花了一小时。而在 MATH 子集上评估这些模型大约花了 3 小时，批量大小 = 16。

为了解决导入错误，我们使用了

A.2 解决一些执行错误

。对于缩进错误，我们作为后处理步骤使用了 Python 标准库中的 `inspect` 模块。

A.3 PAL 提示和 MATH 提示

在我们的实验中，我们采用来自 (?) 的 PAL 提示来评估 LLMs 在 ASDiv 数据上的表现，然而我们仅包含了八个演示中的三个，因为我们观

察到包含整个集没有进一步的提升。完整的提示在图 4 中。

用于评估 LLMs 在 MATH 数据上的提示见图 5

图 ?? 展示了用于将 o3-mini 作为 LLM-Judge 提示的提示。

B 训练集标注指南和标注人员背景

B.1 注释格式

:

每行程序代码的前面都有五个字符，其形式是 TPIR 加空格。

列:

T. 转录

P1-5. 处理 (空白表示“无逻辑”)

I. 推理时间计算

R. 结果收集

B.2 转录

纯转录语句，即从问题中转录数据 (例如数字) 或数据之间的关系 (例如方程)。

纯粹的转录陈述仍然可以包含在问题陈述中明确描述的操作；这些操作不算作处理。

```
""" One number is twice as
large as another. The smaller
number is 3. Find the other
number """
```

```
def solution():
```

T `x = 3`

T R `y = 2*x`

R `return y`

即使模型在推理时使用问题进行计算，但没有纯粹的数据或关系转录，T 也不会被标记：

```
"""
```

```
One number is twice as large
as another. The sum of the
numbers is 12. Find the gcd of
the two numbers
```

```
"""
```

```
def solution():
```

I `x = 4`

I `y = 8`

1 R `z = math.gcd(x, y)`

R `return z`

如果模型执行了部分计算 (例如，将纯转录和真实处理合并)，那么它不会收到 T；它可能会接收到其他标签。

```

"""
Two boards have total length
10; the long board is 2 longer
than the short board. Find the
lengths of the two boards
"""
def solution():
T     total_length = 10
T     difference = 2
2IR   short_length = (total_length -
        difference) / 2
T R   long_length = short_length +
        difference
R     return short_length, long_length

```

B.3 处理类型

评论从未被标记为任何东西!!

1. 概念库

- 稍后调用库函数
- 标记整个函数; 但不标记注释

2. 概念原始

- 以正确的方式使用了原语, 无法通过 lib 进行简化

3. 从零开始的实现

- 大致正确, 看起来像是一个库函数的内联实现
- 为自己编写一个函数, 然后调用 (将调用标记为实现。)

4. 暴力破解

- 遍历所有空间
- 标记整个循环

5. 无组织的

- 可能不正确 (或多或少是随机的?, 无组织的)

空白的“处理类型”字段 (即一个空格 “ ”) 表示没有处理发生。

B.4 推理时计算

如果模型跳过步骤 (无论是完全跳过, 还是部分跳过 (在这种情况下仍然会有一个处理标签)), 那么该行获得“推理时间计算”标志

在完全预计算模型的情况下, 处理类型可能为空, 并且 I 标志将在之后出现。

```

def solution():
I     x = 6 # lcm of 3 and 2
        return x

```

如果模型进行了符号操作以避免使用符号公式库, 则会收到 I 作为处理/转录标签。

B.5 结果收集

在最后一列中, 返回最终答案的行或存储包含最终答案的变量的行被标记为 R。

```

def solution():
IR   x = 6 # lcm of 3 and 2
R     return x

""" One number is twice as
large as another. The sum of
the numbers is 12. Find the
gcd of the two numbers"""
def solution():
I     x = 4
I     y = 8
1 R   z = math.gcd(x, y)
R     return z

```

B.6 标注者背景

生成程序的注释过程耗时且不适合大规模进行, 因为这需要对代码理解有专业知识。两位注释者都是计算机科学领域的专家, 且已经阅读过编程语言和相关主题的资料, 这使他们更适合进行注释工作。

此外, 程序步骤和代码结构模式比自然语言更不容易产生主观性, 携带的细微差别也较少, 这可能导致分歧或偏见。例如, 很难将一个蛮力程序中的库调用误认为其他东西。

C 代码结构判决和 LLM 判决性能的更多结果

C.1 每类 F_1 分数

表 3 提供了两个代码评审的精确度、召回率和 F1 分数。

我们提供了一些程序示例, 其中代码结构判断错误分类了混乱的程序及其在图 6 中的真实标签。

D 对生成程序的进一步分析

D.1 在 MATH500 数据集中生成程序中最常见的错误

StarCoder2 生成了 60 个有错误的程序: 其中 23 个有未定义的符号, 15 个从库中调用未定义的功能。例如, 从数学库中调用 "log3"。Qwen2.5 生成了 40 个有错误的程序, 其中 8 个有未定义的符号, 7 个有对象的未定义属性, 例如, 浮点对象没有属性 denominator。GPT4o-mini 仅生成了 5 个有错误的程序, 其中两个有未定义的符号。

	Precision	Recall	F_1
Conceptual	1.	1.	1.
Primitive	0.72	0.95	0.82
From-scratch Implementation	0.66	1.	0.8
Brute-Force	0.88	1.	0.94
Disorganized	0.5	0.16	0.25
No Logic	0.91	0.68	0.78
	Precision	Recall	F_1
Conceptual	1.	0.88	0.94
Primitive	0.56	0.95	0.71
From-scratch Implementation	0.83	0.83	0.83
Brute-Force	0.83	0.93	0.88
Disorganized	0.6	0.25	0.35
No Logic	0.75	0.37	0.5

Table 3: 代码结构判定器（上）和 LLM 判定器（下）的精确度、召回率和 F_1

D.2 各数学科目的程序分析

图 7 展示了来自 MATH500 数据集中每个数学科目的分类层次分布。

Figure 4: PAL prompt used to prompt evaluated models with three demonstrations on the ASDiv dataset.

System: Write python program to solve math problems.

User: # Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

Assistant:

```
# solution in Python:
def solution():
    """Michael had 58 golf balls. On tuesday, he lost 23 golf balls.
    On wednesday, he lost 2 more. How many golf balls did he have at
    the end of wednesday?"""
    golf_balls_initial = 58
    golf_balls_lost_tuesday = 23
    golf_balls_lost_wednesday = 2
    golf_balls_left = golf_balls_initial \
        - golf_balls_lost_tuesday \
        - golf_balls_lost_wednesday
    result = golf_balls_left
    return result
```

User: # Q: Olivia has \$ 23. She bought five bagels for \$ 3 each. How much money does she have left?

Assistant:

```
# solution in Python:
def solution():
    """Olivia has $23. She bought five bagels for $3 each. How much
    money does she have left?"""
    money_initial = 23
    bagels = 5
    bagel_cost = 3
    money_spent = bagels * bagel_cost
    money_left = money_initial - money_spent
    result = money_left
    return result
```

User: # Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

Assistant:

```
# solution in Python:
def solution():
    """There were nine computers in the server room. Five more
    computers were installed each day, from monday to thursday. How
    many computers are now in the server room?"""
    computers_initial = 9
    computers_per_day = 5
    num_days = 4 # 4 days between monday and thursday
    computers_added = computers_per_day * num_days
    computers_total = computers_initial + computers_added
    result = computers_total
    return result
```

Figure 5: Prompt structure for evaluating on MATH dataset.

```
System: Write python programs to solve math problems
User: # Q: Find the coefficient of  $x^3$  when  $3(x^2 - x^3 + x) + 3(x + 2x^3 - 3x^2 + 3x^5 + x^3) - 5(1 + x - 4x^3 - x^2)$  is simplified.
Assistant:
# solution in Python:
from sympy import symbols, simplify
def solution():
    x = symbols('x')
    expr = 3*(x**2 - x**3 + x) + 3*(x + 2*x**3 - 3*x**2 + 3*x**5 + x**3) -
    5*(1 + x - 4*x**3 - x**2)
    simplified_expr = simplify(expr)
    x3_coefficient = simplified_expr.as_coefficients_dict()[x**3]
    result = x3_coefficient
    return result

User: # Q: The surface area of a sphere with radius  $r$  is  $4\pi r^2$ . Including the area of its circular base, what is the total surface area of a hemisphere with radius 6 cm? Express your answer in terms of pi
Assistant:
# solution in Python:
import math
def solution():
    radius = 6
    # Surface area of the hemisphere
    hemisphere_area = 2 * math.pi * radius**2
    # Area of the circular base
    base_area = math.pi * radius**2
    # Total surface area
    total_surface_area = hemisphere_area + base_area
    result = total_surface_area / math.pi
    return result

User: # Q: Monica tosses a fair 6-sided die. If the roll is a prime number, then she wins that amount of dollars (so that, for example, if she rolls 3, then she wins 3 dollars). If the roll is composite, she wins nothing. Otherwise, she loses 3 dollars. What is the expected value of her winnings on one die toss? Express your answer as a dollar value to the nearest cent.
Assistant:
# solution in Python:
def solution():
    # Probabilities of each outcome
    prime_prob = 1 / 6
    composite_prob = 1 / 3
    otherwise_prob = 1 / 6
    # Expected value of each outcome
    prime_expected_value = (2 * prime_prob) + (3 * prime_prob) + (5 * prime_prob)
    composite_expected_value = 0 * composite_prob
    otherwise_expected_value = -3 * otherwise_prob
    # Total expected value
    total_expected_value = prime_expected_value + composite_expected_value + otherwise_expected_value
    # Dollar value to the nearest cent
    result = "{:.2f}".format(total_expected_value)
    return result
```

```

def find_other_number():
    difference = 100
    one_number = 91
    other_number = one_number + difference
    return other_number
result = find_other_number()

True label: "Primitive"

def smallest_page_number():
    # The least common multiple (LCM) of 4 and 13
    lcm = 4 * 13
    # Since 4 and 13 are co-prime
    return lcm
result = smallest_page_number()

True label: "No Logic"

```

Figure 6: 被编码结构评判员标记为“不整齐”的实例及其真实标签。

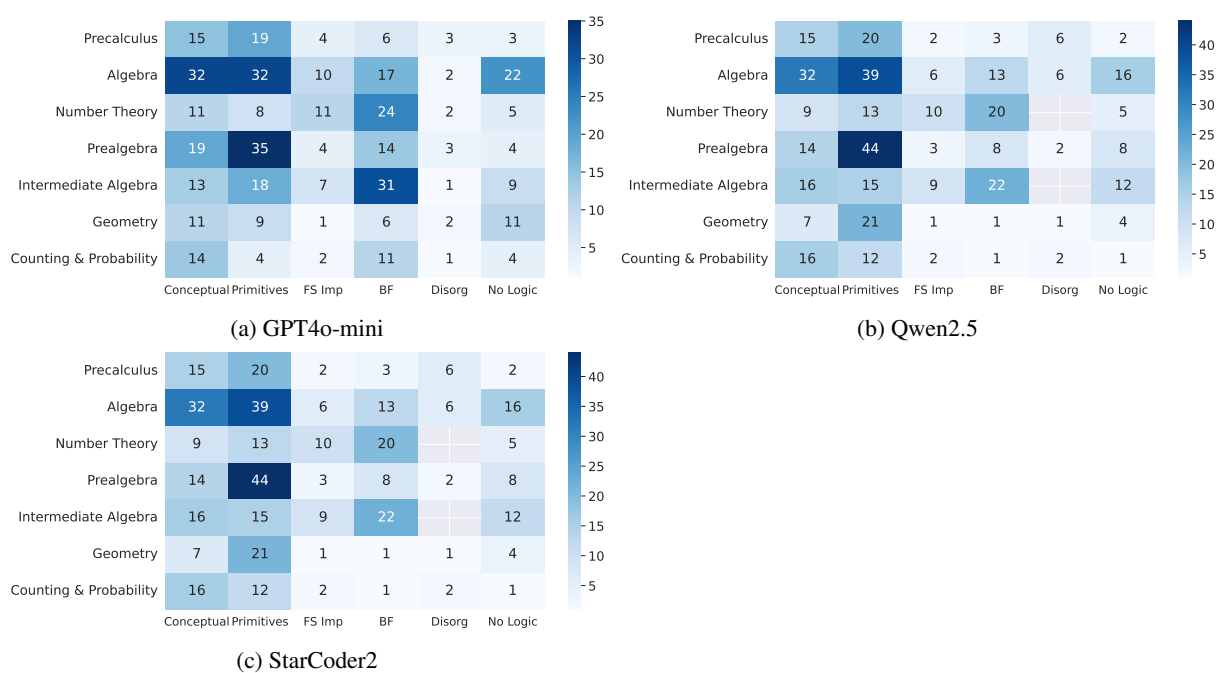


Figure 7: 从 MATH500 数据集中每个数学学科的分类计数分布用于评估模型。类别: 'FS Imp': 从零开始的实现, 'BF': 暴力破解, 'Disorg': 无组织。