正确是不够的：在训练用于数学推理的大型语言模型中的结果监控陷阱

Jiaxing Guo[1,2], **Wenjie Yang**[1,2], **Shengzhong Zhang**[3], **Tongshan Xu**[1]
Lun Du[2], Da Zheng[2], Zengfeng Huang[1]

[1] School of Data Science, Fudan University [2] Ant Group
[3] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics

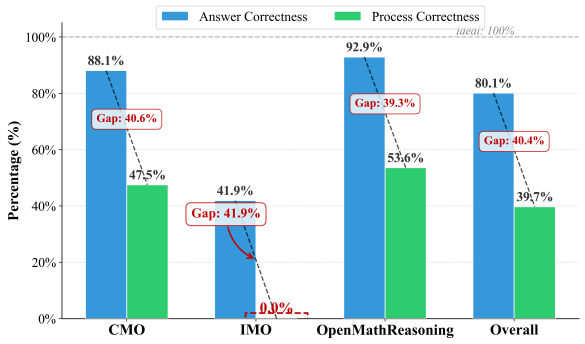

Figure 1: 大型语言模型经常通过基本上不可靠的推理过程来获得正确的数学答案。对 MATH-OLYMPIADEVAL 的人类评估显示，高答案正确性与低过程正确性之间存在显著差异。

Abstract

结果奖励大语言模型（LLMs）在数学问题解决方面表现出显著的成功。然而，这种成功往往掩盖了一个关键问题：模型通常通过基本上不健全的推理过程来获得正确答案，这种现象表明了奖励滥用。我们引入了 MathOlympiadEval，这是一个带有细粒度注释的新数据集，它揭示了 LLMs 的答案正确性与其低推理过程正确性之间的显著差距。现有的自动化方法，如 LLM-as-a-judge，难以可靠地检测这些推理缺陷。为了解决这个问题，我们提出了 ParaStepVerifier，这是一种用于对数学解决方案进行细致的、逐步验证的新方法。ParaStepVerifier 可以识别出不正确的推理步骤。实验证明，与基线相比，ParaStepVerifier 在识别有缺陷的解决方案方面显著提高了准确性，特别是在复杂的多步骤问题中。这为评估和训练具有真正数学推理能力的 LLMs 提供了一条更稳健的路径。

1 介绍

复杂的数学推理代表了人工智能中的一个关键前沿领域，在这一领域中，严谨的评估方法对于实现有意义的进展至关重要。大型语言模型（LLMs）越来越多地被用于数学问题解决，通常通过监督微调（SFT）和强化学习（RL）进行训练，以优化最终结果 (Liu et al., 2024; Guo et al., 2025) 。例如，依赖于基于结果的奖励优化的模型如 DeepSeek-R1 (Guo et al., 2025) 和 QwQ-32B (Qwen, 2025) 在数学基准测试中取得了令人印象深刻的性能。这些成功使得一些人建议，仅关注结果的优化可能就足够了，这有可能解决长期存在的基于结果和基于过程的奖励策略之间的争论。

然而，我们对 MATHOLYMPIADEVAL 上 LLM 输出的细致人工评估——这是我们构建的一个多样化的数据集，包含具有挑战性的数学问题，并对 LLM 生成的解决方案进行最终答案正确性和细粒度推理有效性注释——揭示了一个关键的脱节：模型经常通过根本上错误的推理过程得出正确的最终答案。具体而言，尽管像 DeepSeek-R1 和 QwQ-32B 这样的面向推理的模型在% 答案准确率上达到 80.1，人工专家判断其中只有 39.7 % 的相应推理路径是合理的。换句话说，在基于最终答案被认为“正确”的问题中，经过推理有效性仔细审查后，只有不到一半是确实正确的（图 1 ）。这种现象不仅限于问题的一个狭窄子集，而是广泛存在于从主要数学竞赛获得的任务中，如国际数学奥林匹克（IMO）、中国数学奥林匹克（CMO）以及社区驱动的平台如问题解决艺术（AoPS）。在这些不同的来源中，LLMs 始终表现出一种模式，即在逻辑上不健全的推理基础上产生看似正确的答案。

这样的发现强烈表明，仅仅为了最终结果优化可以激励模型采用“捷径”策略——有效地进行奖励作弊——学习如何最大化结果指标而不发展真正的推理完整性。这个问题进一步加剧了系统性传播的风险。奖励作弊症状的缺陷解决过程可能会在用于监督微调（SFT）的数据集中出现，可能会在后续模型生成中延续不良的推理模式。我们对据报道用于 SFT 流程的 OpenMathReasoning (Moshkov et al., 2025) 样本的分析表明，57.7 % 的解决痕迹包含推理缺陷。这引发了严重的担忧，即模型错误输出污染训练数据的反馈循环，增强了肤浅或错误的推理，最终限制了模型可靠解决数学问题的能

力。

这些结果表明，现有的基准无法充分评估大型语言模型（LLMs）的推理能力，因为仅依赖结果的指标通常无法捕捉到关键的过程层面的缺陷。作为裁判的大规模 LLM 方法——尽管被越来越多地采用用于自动评估——面临着重大的可靠性挑战（Faggioli et al., 2023; Li et al., 2024）。这些自动裁判存在一系列问题，包括提示敏感性和不一致性（Sclar et al., 2024; Ganesh et al., 2025），以及风格偏差和自我偏好（Zheng et al., 2023; Panickssery et al., 2024）——最终显示出与人类判断较弱的相关性（Zheng et al., 2023; Xie et al., 2025）。

因此，尽管结果奖励模型取得了明显的成功，仍然迫切需要评估可以对推理过程本身进行评估的方法。超越仅仅检查答案是有效检测和惩罚奖励作弊行为的关键。为弥补这一差距，我们引入了 ParaStepVerifier——一个专为数学解决方案的细致、逐步验证而设计的智能体。ParaStepVerifier 系统地识别出错误的推理，精确定位逻辑崩溃的具体步骤。这种细致的反馈不仅能够对故障模式进行详细的诊断，还有助于整理更高质量的训练数据。通过提供这种结构化的见解，ParaStepVerifier 为模型评估和发展更为稳健的 LLM 数学推理提供了更可靠的基础。

我们的主要贡献是：

- 我们构建了 MathOlympiadEval，这是一个包含来自 CMO、IMO 和 OpenMathReasoning 的具有挑战性的数学问题的多样化数据集。每个问题都由人类专家标注以验证推理的正确性。
- 在进行数学推理时，我们发现了结果奖励的 LLMs（例如，DeepSeek-R1）在答案准确性（80.1 %）和推理正确性（39.7 %）之间存在显著差距，这表明正确答案常常源于错误的推理。我们进一步识别出常见的失败模式，例如猜测和逻辑错误，并显示 57.7 % 的采样 SFT 数据（例如，OpenMathReasoning）中包含这种缺陷，这引发了对奖励欺骗和因数据污染导致推理退化的担忧。
- 我们提出了 ParaStepVerifier，这是一种执行逐步验证和详细错误识别的代理。它可以评估数学推理过程，检测奖励欺骗，并适用于解答和基于证明的任务。

2 相关工作

大语言模型作为评判者 现有的基准测试未能充分评估大型语言模型（LLMs）的推理过程，因为仅基于结果的指标往往忽视关键的过程层面缺陷。虽然越来越多地使用可扩展的 LLM-作为-评审方法，但其面临显著的可靠性挑战（Faggioli et al., 2023; Li et al., 2024）。这些自动化评审存在持续的问题，包括：不一致性和提示敏感性（Sclar et al., 2024; Ganesh et al., 2025）；对冗长性或特定风格的固有偏见，以及偏袒与自身风格相似的输出（Zheng et al., 2023; Panickssery et al., 2024）；倾向于忽视逻辑谬误（Wang et al., 2024; Zheng et al., 2023）；与人类判断相关性差（Zheng et al., 2023; Xie et al., 2025）；以及当评审与被评估模型存在相同弱点时，无法检测错误（Wang et al., 2024; OpenAI et al., 2024）。

推理模型 在评估大型语言模型的数学推理方面存在关键差距。根据与 Petrov et al. (2025) 关于大型语言模型证明困难及不可靠的“大型语言模型作为评判者”的评估结果，我们的 MATHOLYMPIADEVAL 数据集广泛记录了答案过程正确性之间的差异。为了解决这个问题，ParaStepVerifier 提供了强有力的逐步验证。类似地，Mahdavi et al. (2025) 识别了奥林匹克问题解决中的奖励作弊行为；MATHOLYMPIADEVAL 扩展了这一发现，展示了在各种数学竞赛中，包括高中水平（表格 1），这一问题的普遍性。

我们的研究还质疑了对训练验证器（例如 Heimdall (Shi and Jin, 2025)）依靠基于结果的奖励，因为 MATHOLYMPIADEVAL 显示这种监督掩盖了潜在的推理缺陷。尽管 Yang et al. (2025) 的 StepMathAgent 提出了一种逐步评分系统，但我们的 ParaStepVerifier 专注于明确地确认每个推理步骤的二元正确性。

3 MATHOLYMPIADEVAL：用于数学推理及其评估的数据集

本节介绍了 MATHOLYMPIADEVAL，一个新的数据集，旨在促进对大型语言模型（LLMs）中数学推理的细致评估。我们首先定义数据集所涉及的核心推理任务，然后详细介绍其构建和严格的标注过程。最后，我们展示使用 MATHOLYMPIADEVAL 获得的初步评估结果，突出了模型答案正确性与其推理合理性之间的重要差异。

3.1 任务定义

给定一个来自 MATHOLYMPIADEVAL 数据集的数学问题 P 和一个由 LLM 生成的候选解决方案 S，其中 S 由一系列推理步骤 $\{s_1, s_2, \ldots, s_n\}$ 组成，最终得到答案 A，评估任务旨在评估 S 的两个不同方面：

1. 答案正确性（$C_A \in \{0,1\}$）：对于找答案问题，我们确定最终答案 A 是否与参考答案 R 匹配。这个二元判断代表了通常在数学基准中使用的传统结果导向评估。

2. 推理正确性（$C_R \in \{0,1\}$）：我们评估解题过程是否遵循有效的数学原理，没有逻辑缺陷、不正确的运算或不合理的结论。只有当每一步 s_i 从问题陈述和前面的步骤 $(P, \{s_1, \ldots, s_{i-1}\})$ 逻辑上推导出时，解决方案才具有正确的推理。

3.2 MATHOLYMPIADEVAL 的构建与注释

问题策划 MATHOLYMPIADEVAL 数据集由 204 个高质量的数学问题组成，这些问题来自四个主要类别：中国数学奥林匹克初赛 (CMO-Preliminary; 86 个问题)、国际数学奥林匹克 (IMO; 62 个问题) 和 NVIDIA 的 OpenMathReasoning 数据集 (56 个问题) (Moshkov et al., 2025)。为了确保问题的多样性并减少选择偏差，CMO-Preliminary、IMO 和 OpenMathReasoning (特别是来自《解题艺术》"高中奥林匹克"部分) 的问题是随机抽取的。经过精选的数据集包括 58 个证明题和 146 个求解题，这代表了竞赛级别数学中的一系列典型挑战。

针对 MATHOLYMPIADEVAL 中的问题生成的候选解决方案使用了两个先进的开源 LLM：QwQ-32B 和 DeepSeek-R1，这两个模型都通过强化学习和基于结果的奖励进行了微调。一个一致的提示方法确保模型输出包含完整的推理步骤。对于基于证明的问题，模型生成了正式的证明和推理过程。对于寻找答案的问题，模型提供了最终答案和详细的解决路径。这些生成的解决方案与问题一起构成了数据集的一部分。

标注方法 在 MATHOLYMPIADEVAL 中，LLM 生成的解答经过严格的注释过程。首先，在适用的情况下，进行初步的自动验证以检查答案的正确性。随后，由具有丰富数学竞赛经验的数学研究生进行人工注释阶段。注释者会得到初步验证结果和官方参考解答。所有注释者都接受了统一的培训和指导。

为了深入了解模型的缺陷，在推理中被识别为不正确的解决方案被系统地分类。这个错误分类是数据集标注的一个关键特征。预定义的错误类型包括：

- 通过猜测解决：检测那些依赖于特定实例、通过发现模式而没有严谨推广或其他非演绎跳跃来得出结论的实例。

- 循环推理：识别在前提中隐含或明确假设了待证明命题的论证。

- 不等式操作错误：检测不正确应用或转换不等式性质和运算的情况。

- 计算错误：识别解题步骤中的算术或代数错误。

- 逻辑谬误: 检测不属于其他类别的更广泛的错误推理模式或形式/非形式谬误。

包括错误分类在内的每个标注都经过了三个回合的迭代审核过程，每个回合由不同的专家注释员进行，以确保 MATHOLYMPIADEVAL 中标签的可靠性和准确性。

3.3 对 MATHOLYMPIADEVAL 的 LLM 推理的初步评估

我们对 MATHOLYMPIADEVAL 数据集的初步评估揭示了当前大型语言模型中的一个关键现象：通过基于结果的奖励进行训练的模型经常通过有缺陷或不合理的推理过程达到正确的最终答案。表格 1 展示了一个统一分析，比较了来自 MATHOLYMPIADEVAL 的 146 个答案找到问题的答案正确性与人类验证的推理正确性。

Metric	CMO	IMO	OpenMathReasoning	Overall
Total Problems (Answer-Finding)	59	31	56	146
Answer Correctness (%)	88.1	41.9	92.9	80.1
Human Correctness (Reasoning) (%)	47.5	0.0	53.6	39.7
Correct Answers (Count)	52	13	52	117
Sound Reasoning Among Correct Answers (%)	53.8	0.0	57.7	49.6

Table 1: 对答案正确性与人工验证推理正确性之间的统一分析，以及在最初 146 个答题问题的正确答案中合理推理的比率。

如表 1 所示，大语言模型在这些问题上的总答案正确率很高。然而，底层推理过程的人类验证正确性要低得多。这个鲜明的差异表明，即使在生成正确的最终答案时，模型也经常采用有缺陷或不完整的推理。这个问题在 MATHOLYMPIADEVAL 内的具有挑战性的国际数学奥林匹克问题中尤为显著。虽然模型实现了显著的答案正确率，但这些正确答案中没有一个来源于健全的推理过程。对于 CMO-Preliminary 和 OpenMathReasoning 问题，推理正确率较高，但相当部分正确回答的问题仍然缺乏有效的支持性推理步骤。

4 ParaStepVerifier 方法论

我们引入了 ParaStepVerifier，一种用于评估数学推理的新型自动化方法。它验证提议解决方案中每一步的逻辑正确性。这种方法使用大型语言模型（LLMs）作为验证代理，能够进行更详细、可解释且可能更准确的评估，相较于整体或仅结果的方法。

4.1 逐步验证任务

核心任务是评估候选解 S 对于给定数学问题 P 的正确性。我们将 S 视为一个有序的推理步骤序列 $S = \{s_1, s_2, \ldots, s_n\}$。整个解决方案的有效性取决于每个步骤的正确性。

4.1.1 验证背景和目标

每个推理步骤 $s_i \in S$ 都在特定的上下文中进行评估。步骤 s_i 的主要验证上下文 C_i 包括问题陈述 P 和前面有效步骤的历史 $H_{i-1} = \{s_1, s_2, \ldots, s_{i-1}\}$。目标是确定 s_i 是否逻辑上源于这个已建立的上下文 (P, H_{i-1})。

若一个步骤 s_i 包含一个或多个推理错误，则该步骤

4.1.2 错误分类

是不正确的。第 3.2 节详细介绍了注释过程 $\mathcal{E}$ 下关键错误类型的集合。值得注意的是，“计算错误”被排除在 ParaStepVerifier 的验证提示之外，因为函数调用通常可以解决此类错误。此外，我们的数据集将“计算错误”定义为包含证明步骤中的计算错误，而不仅仅是在寻求答案的问题中。

4.1.3 形式验证定义

令 $m = |\mathcal{E}|$ 为错误类型的数量。对于每个错误类型 $j \in \{1, \ldots, m\}$，错误检测函数 $e_j(P, H_{i-1}, s_i)$ 如果在从上下文中推导出 s_i 时出现错误类型 j，则返回 1，否则返回 0。步骤 s_i 的验证函数 V 确定其正确性：

$$V(s_i|P, H_{i-1}) = \begin{cases} 1 & \text{if } \sum_{j=1}^{m} e_j(P, H_{i-1}, s_i) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

值为 1 表示步骤正确（错误和为 0），值为 0 表示至少有一个错误。这可以简洁地表示为：

$$V(s_i|P, H_{i-1}) = \prod_{j=1}^{m} (1 - e_j(P, H_{i-1}, s_i)) \tag{2}$$

当且仅当所有步骤都正确时，整体解 S 才是正确的：

$$\text{Correctness}(S|P) = \prod_{i=1}^{n} V(s_i|P, H_{i-1}) \tag{3}$$

4.2 基于 LLM 的实现：ParaStepVerifier 代理

我们使用基于 LLM 的代理来实现逐步验证。该代理在每个步骤的上下文中进行分析，以识别错误并提供判断。

4.2.1 输入准备：解分解

我们的系统自动将数学解答分割成离散步骤，利用现代推理模型固有的逐步结构。一个解答 S 被分解成一个有序的推理步骤序列 $S = \{s_1, s_2, \ldots, s_n\}$，其中每个 s_i 是一个连贯的数学推理单元。为了确保每个步骤足够实质性以便进行评估，如果 s_k 的长度低于经验设定为 12 个标记的阈值 θ，则相邻步骤 s_k 和 s_{k+1} 被连接在一起。

要验证步骤 s_i，LLM 代理会收到一个精心构建的上下文。这个上下文包括原始问题 P、前面的步骤 $H_{i-1} = \{s_1, s_2, \ldots, s_{i-1}\}$、当前的步骤 s_i 以及任何未来的步骤 $F_{i+1} = \{s_{i+1}, \ldots, s_n\}$。用于验证 s_i 的完整上下文是一个称为分析上下文的元组 $(P, H_{i-1}, s_i, F_{i+1})$。虽然未来的步骤 F_{i+1} 有助于 LLM 理解解决方案的走向，但代理的主要任务是根据从验证上下文 (P, H_{i-1}) 逻辑推导的严格性来判断 s_i 的正确性，并以我们的提示结构（B 节）为指导。

4.2.2 自适应验证策略

ParaStepVerifier 采用一种基于解题步骤数量的自适应策略，以提高评估的效率和准确性。对于只有单个推理步骤的简洁解答，系统使用结合 LLM 评估和针对性错误检测的直接方法，以快速验证解答。对于复杂的多步解答，ParaStepVerifier 启动其完整的逐步验证机制。它仔细分析每一个步骤 s_i，使用之前验证过的步骤序列 H_{i-1} 和后续步骤 F_{i+1} 作为上下文。

4.3 系统架构及并行化

为了有效处理长解答，ParaStepVerifier 在并行处理架构中实现。主要组件包括：解答分解模块：实现步骤分割和连接。任务队列：包含验证任务，每个对应一个步骤 s_i 及其上下文。并行验证工作者：一个由工具如 `concurrent.futures.ProcessPoolExecutor` 管理的进程池。每个工作者都执行一个任务，构建提示，查询 LLM，并解析判断。资源管理：根据可用资源自适应管理工作者数量。使用 LLM API 进行批处理以提高效率。结果聚合：收集所有步骤的真假判断，通过方程 3 确定整体解答正确性。这种并行架构显著减少了复杂数学解答的评估时间。

5 实验

5.1 设置

数据集和基线 我们在我们的数据库 MATH-OLYMPIADEVAL 上评估我们的方法。我们评估了几个在数学推理任务中表现出显著能力的

模型的性能，包括 Gemini-2.0-flash, QwQ-32B, o1-mini, o3-mini, Qwen3-32B, o1 和 Gemini-2.5-Pro。这些模型作为数学证明或解决方案的基线验证代理。

为了评估我们的验证模型在识别有缺陷的数学解决方案或证明方面的性能，我们的主要评估指标是 F1 分数。我们将此任务框定为一个二分类问题，其中“正”类表示给定的解决方案是不正确的。

令 $\mathcal{S}_{\text{incorrect}}^{\text{GT}}$ 表示在真实标签中被标记为不正确的解集，$\mathcal{S}_{\text{incorrect}}^{\text{Pred}}$ 表示被我们的模型识别为不正确的解集。真正正例（TP）、假正例（FP）、假负例（FN）、精度、召回率和 F1 分数定义如下：

$$\text{TP} = |\mathcal{S}_{\text{incorrect}}^{\text{GT}} \cap \mathcal{S}_{\text{incorrect}}^{\text{Pred}}|, \tag{4}$$

$$\text{FP} = |\mathcal{S}_{\text{incorrect}}^{\text{Pred}} \setminus \mathcal{S}_{\text{incorrect}}^{\text{GT}}|, \tag{5}$$

$$\text{FN} = |\mathcal{S}_{\text{incorrect}}^{\text{GT}} \setminus \mathcal{S}_{\text{incorrect}}^{\text{Pred}}|, \tag{6}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{7}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{8}$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{9}$$

5.2 结果

本节详细介绍了 ParaStepVerifier 的实证表现。我们首先展示了它在提高数学推理验证准确性方面的整体有效性及其相对成本效益。然后，我们分析了其在评估具有扩展推理链的解决方案时的稳健性。

5.2.1 整体验证性能和成本效益

我们的评估结果在表格 2 中进行了总结。结果显示，与各种基础模型中的标准大型语言模型作为评判基线相比，ParaStepVerifier 显著提高了识别不正确数学解决方案的 F1 分数。例如，即使是一个较为强大的基础模型如 Gemini-2.0-flash，当结合 ParaStepVerifier 时，其总体 F1 分数从 75.89 % 提升到 83.39 %。这突显了 ParaStepVerifier 通过其结构化的、逐步验证的过程来指导基础模型实现更准确的评估能力，在各种不同的数学子数据集上均观察到了一致的性能提升。

ParaStepVerifier 使基本模型能够实现甚至超越更强大、最先进模型（如 o1）的评估能力，同时与基本模型结合的 ParaStepVerifier 在成本效益上比强大模型更有优势。这里，强大模型仅用于一个简单的“LLM-as-a-judge”功能。如表 2 所详细说明，ParaStepVerifier 与 Gemini-2.0-flash 结合实现了最佳性能（总体 F1：83.39 %），并且在代理解决方案中也是最具成本效益的。它略优于 Gemini2.5Pro（总体 F1：83.26 %），后者作为最佳“LLM-as-a-judge”解决方案的估计成本为 $ 0.97，远低于 Gemini2.5Pro 的 $ 9.32 或 o3-mini 作为“LLM-as-a-judge”时的 $ 4.12，尽管后两者的 F1 评分较低。这表明 ParaStepVerifier 提供了一种有效的路径，以显著提高数学推理评估的准确性，并且通常具有更高的成本效益。

虽然 ParaStepVerifier 在测试的模型中显示了显著的改进，但我们目前并没有展示将 ParaStepVerifier 与列在 LLM-as-a-judge 类别中的最强大 LLM（如 o1 或 Gemini2.5Pro）整合的结果。这主要是因为 ParaStepVerifier 固有的详尽的、逐步验证过程所需的巨大计算成本和资源需求。每次使用这种方法进行解决方案验证时都涉及多次 LLM 调用，将其应用于最大模型将导致显著更高的费用和更长的处理时间，这超出了当前实验设置的范围。未来的工作可能会在资源允许和模型 API 成本变化时探索这些配置。

Model	F1 Score (%)				Cost ($)
	CMO	IMO	OpenMathReasoning	Overall	
LLM-as-a-judge					
QwQ-32B	43.33	65.22	37.84	52.91	0.84
Qwen3-32B	51.61	78.43	40.00	62.75	1.04
Gemini-2.0-flash	57.14	94.92	62.75	75.89	0.37
o1-mini	48.57	77.23	41.86	60.75	4.10
o3-mini	67.47	95.80	69.09	80.93	4.12
o1	65.71	94.02	20.00	74.65	56.21
Gemini2.5Pro	70.89	97.52	70.37	83.26	9.32
ParaStepVerifier					
ParaStepVerifier_o1-mini	68.89	77.23	64.00	71.37	10.64
ParaStepVerifier_QwQ-32B	61.73	90.27	58.82	74.29	2.18
ParaStepVerifier_Qwen3-32B	72.09	94.92	72.73	82.63	2.71
ParaStepVerifier_Gemini-2.0-flash	69.57	97.52	75.86	83.39	0.97

Table 2: 在 MATHOLYMPIADEVAL 上的评估结果。我们在 MATHOLYMPIADEVAL 数据集上报告 F1 分数（%）和估计的评估成本（$）。

5.2.2 关于具有扩展推理链的解决方案的性能

为了评估 ParaStepVerifier 在更复杂问题上的有效性，我们将其与 LLM+EC（作为判断者的 LLM 与错误分类）策略在具有扩展推理链的解决方案中的表现进行比较。表 3 详细显示了这种比较，突出显示了 ParaStepVerifier 相对于 LLM+EC 在 F1 分数上取得的相对提升。

表 3 中的结果表明，随着解决方案复杂性的增加（步骤数 > 7 和 > 9），ParaStepVerifier 均能持续实现明显的相对 F1 分数提升。例如，以 QwQ-32B 作为基础模型时，ParaStepVerifier 在步骤超过 7 的解决方案中实现了 17.49 % 的相对 F1 分数提升，对于超过 9 步的解决方案则显著提升到 28.56 %，相比于 LLM+EC。类似的显著相对增长在其他基础模型中也有观察到；o1-mini 在最复杂问题（>9 步）中记录了令人瞩目的 50.02 % 的相对提升。即使对于

Base Model	Verification Strategy	F1 (%) (Step >7)	F1 (%) (Step >9)
QwQ-32B	LLM+EC	72.22	66.67
	ParaStepVerifier	84.85	85.71
	Rel. Improv. (% Δ)	+17.49 %	+28.56 %
o1-mini	LLM+EC	60.61	44.44
	ParaStepVerifier	72.22	66.67
	Rel. Improv. (% Δ)	+19.15 %	+50.02 %
Qwen3-32B	LLM+EC	80.00	85.71
	ParaStepVerifier	87.18	88.89
	Rel. Improv. (% Δ)	+8.98 %	+3.71 %
Gemini-2.0-flash	LLM+EC	83.33	66.67
	ParaStepVerifier	85.00	85.71
	Rel. Improv. (% Δ)	+2.00 %	+28.56 %

Table 3: 在扩展推理链上的验证策略之间的 F1 性能比较。每个模型的专用行显示 F1 分数（绿色粗体值）中由 ParaStepVerifier 相对于 LLM+EC 实现的相对提高（Rel. Improv. % Δ）。

像 Qwen3-32B 和 Gemini-2.0-flash 这样更强的基线，ParaStepVerifier 也提供了明确的优越性，其中 Gemini-2.0-flash 在超过 9 步的解决方案中显示出 28.56 % 的相对 F1 增加。

这种显著的正相对改进的一致趋势，如表格 3 中每个模型的 Rel. Improv. (% Δ) 行所详细描述的，强调了 ParaStepVerifier 在准确验证需要广泛、多步骤推理的解决方案方面的增强能力。

6 消融实验

在本节中，我们进行消融研究以评估我们的 ParaStepVerifier 流程的关键方面。首先，在第 6.1 节中，我们分析错误分类和逐步验证的单独和组合影响，这些是评估数学推理正确性的核心机制。随后，在第 6.2 节中，我们检验并行验证在提高处理效率方面相较于顺序方法的有效性。

6.1 错误分类与逐步验证

Base Model	Verification Strategy	CMO	IMO	OpenMathReasoning	Overall
QwQ-32B	LLM-as-a-judge	43.33	65.22	37.84	52.91
	ParaStepVerifier w/o EC	47.76	70.83	42.86	57.56
	LLM+EC	55.07	81.90	50.00	66.67
	ParaStepVerifier	61.73	90.27	58.82	74.29
o1-mini	LLM-as-a-judge	48.57	77.23	41.86	60.75
	ParaStepVerifier w/o EC	58.54	86.24	44.44	68.64
	LLM+EC	58.97	70.83	63.83	65.16
	ParaStepVerifier	68.89	77.23	64.00	71.37
Qwen3-32B	LLM-as-a-judge	51.61	78.43	40.00	62.75
	ParaStepVerifier w/o EC	58.33	88.29	52.17	71.62
	LLM+EC	65.85	90.27	65.31	77.05
	ParaStepVerifier	72.09	94.92	72.73	82.63
Gemini-2.0-flash	LLM-as-a-judge	57.14	94.92	62.75	75.89
	ParaStepVerifier w/o EC	59.26	90.27	60.38	73.68
	LLM+EC	69.57	97.52	73.33	82.78
	ParaStepVerifier	69.57	97.52	75.86	83.39

Table 4: 关于错误分类和逐步验证影响的消融研究。F1 分数报告针对我们数学推理数据集 MATH-OLYMPIADEVAL 使用不同验证策略的结果。

本节分析了错误分类（EC）和逐步验证（Step-by-Step Verification）对 ParaStepVerifier 的影响，这两个组件是其关键部分。表 4 给出了 F1 分数，展示了它们各自和组合在一起的贡献。

错误分类 错误分类（EC）模块一致地增强了验证性能。首先，将 EC 添加到标准的 LLM-as-a-judge 方法中形成 LLM+EC 可以提高所有基础模型的整体 F1 分数。例如，QwQ-32B 和 Qwen3-32B 显示了显著的提升。其次，EC 还提升了我们的逐步方法：具有 EC 的完整 ParaStepVerifier 在所有模型上都优于没有 EC 的 ParaStepVerifier。这些结果表明，EC 通过实现更精确的判断来优化验证过程。

逐步验证 步骤验证相比于整体评估也提供了明显的优势。即使没有 EC，ParaStepVerifier w/o EC 通常也能比以 LLM 作为评判基准的方法取得更高的整体 F1 分数，比如在像 QwQ-32B 和 o1-mini 这样的模型中可以看到。这突显出了将验证任务分解的固有优势。当整合了 EC 后，逐步进行的方法的优越性变得更为明显。完整的 ParaStepVerifier 一贯超越 LLM+EC，进一步提升了所有基础模型的整体 F1 分数。

6.2 并行验证

为了评估我们的并行处理架构在效率提升方面的效果，我们分析了不同验证策略的运行时间，详细信息见表格 5 。该表比较了 ParaStepVerifier、其顺序对应版本 SeqStepVerifier，以及 LLM-as-a-judge 基线。

Verification Strategy	QwQ-32B	Qwen3-32B	o1-mini	Gemini-2.0-flash
LLM-as-a-judge	8.04	6.61	1.22	0.40
SeqStepVerifier (Sequential)	23.78	19.36	4.05	1.11
ParaStepVerifier (Parallel)	9.88	7.48	1.88	0.51

Table 5: 并行验证策略的运行时间分析（小时）

数据显示，并行化显著减少了验证时间。对于所有测试的模型，相比于 SeqStepVerifier，ParaStepVerifier 显著降低了运行时间，例如，在 QwQ-32B 上实现了超过 2.4 倍的加速。这种在各个模型上的一致加速表明了 ParaStepVerifier 并行架构的有效性，这对于及时处理复杂的数学解决方案至关重要。

虽然 ParaStepVerifier 本质上由于其详细的逐步分析导致比 LLM-as-a-judge 基线具有更高的计算成本，但并行化使其运行时间与该基线相当。因此，ParaStepVerifier 的并行策略有效地减轻了与细粒度、逐步分析相关的计算开销，提供了一种实用且可扩展的解决方案，用于稳健的数学推理评估，同时实现比 LLM-as-a-judge 更为彻底的评估。

7 结论

本文研究了人工智能发展中的一个重大挑战：大型语言模型在数学推理任务中通过基本上有缺陷的过程得出正确答案的倾向。这个问题通常被优先考虑最终结果而不是推理步骤完整性的评估方法所掩盖。我们的工作实证性地确认了当代大型语言模型中的表面答案准确性与真正过程合理性之间的重大脱节，通常将此归因于“奖励漏洞”，即模型利用评估指标而没有形成真正理解，从而导致常见的错误，如无根据的猜测或显著的逻辑缺陷。

为了解决这一问题，我们开发并验证了 ParaStepVerifier，这是一种采用基于代理系统的全新方法，进行详细的逐步验证数学解题的正确性。通过使用我们新构建的 MathOlympiadEval 数据集进行全面评估——这一数据集包含具有详细人类正确性注释的多样化挑战数学问题——我们展示了 ParaStepVerifier 显著提高了数学推理评估的准确性和可靠性。我们的研究结果表明，当应用 ParaStepVerifier 时，在识别错误解答上有持续的改善，超过了传统的整体评估方法，特别是在处理需要延长推理链的复杂问题时。这种提升的性能通常是在更具成本效益的情况下实现的。对于 ParaStepVerifier 组件的进一步分析确认了其逐步分析以及集成错误分类能力的独特优势。

ParaStepVerifier 提供了一个更为稳健和透明的框架，用于审查大型语言模型的推理路径。它作为一种有价值的工具，可以用于更可靠的模型评估，用于指导创建更高质量的训练数据集，并最终促进拥有更可靠且可验证的数学推理能力的人工智能系统的发展。虽然本研究奠定了坚实的基础，但未来的探索可能涉及随着计算能力的进步，将 ParaStepVerifier 与最先进的大型语言模型相结合，以及适应其原则来评估其他专业领域的复杂推理。

8

局限性

虽然 ParaStepVerifier 在更可靠的数学推理评估方面提供了重要的进展，但我们承认存在某些局限性。

我们的发现主要基于 MATHOLYMPIADEVAL 数据集，该数据集强调结构化的数学问题，包括解答问题和基于证明的任务。虽然在这些数学任务上的表现令人鼓舞，但在来自显著不同数学领域的问题（例如，高度抽象的拓扑学、量子场论推导）或具有独特结构呈现的问题上的效果仍需进一步验证。此外，鉴于 ParaStepVerifier 在验证数学证明中的逻辑步骤的能力，其在更广泛的文本推理领域（如评估法律文本或科学论文中论点的连贯性）中的推广潜力是未来研究中一个引人注目的领域，但当前研究尚未证实这种潜力。

依赖于基础 LLM 能力。 ParaStepVerifier 的有效性本质上与它作为验证器所使用的 LLM 的推理和验证能力相关联。虽然我们的框架引导 LLM 进行更准确的评估，但所选基本模型的固有限制或偏见仍可能影响验证结果，尤其是在新颖或极其复杂的推理情况下。由于当前的计算成本限制，使用最大规模的最先进 LLM 作为验证代理进行全面测试受到限制。

References

Guglielmo Faggioli, Laura Dietz, Charles L. A. Clarke, Gianluca Demartini, Matthias Hagen, Claudia Hauff, Noriko Kando, Evangelos Kanoulas, Martin Potthast, Benno Stein, and Henning Wachsmuth. 2023. Perspectives on large language models for relevance judgment. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR ' 23, page 39–50. ACM.

Prakhar Ganesh, Reza Shokri, and Golnoosh Farnadi. 2025. Rethinking hallucinations: Correctness, consistency, and prompt multiplicity.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024. Llms-as-judges: A comprehensive survey on llm-based evaluation methods. *Preprint*, arXiv:2412.05579.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Hamed Mahdavi, Alireza Hashemi, Majid Daliri, Pegah Mohammadipour, Alireza Farhadi, Samira Malek, Yekta Yazdanifard, Amir Khasahmadi, and Vasant Honavar. 2025. Brains vs. bytes: Evaluating llm proficiency in olympiad mathematics. *Preprint*, arXiv:2504.01995.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. 2025. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *Preprint*, arXiv:2504.16891.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. 2024. Llm evaluators recognize and favor their own generations. *Preprint*, arXiv:2404.13076.

Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunovi, Nikola Jovanovi, and Martin Vechev. 2025. Proof or bluff? evaluating llms on 2025 usa math olympiad. *Preprint*, arXiv:2503.21934.

Team Qwen. 2025. Qwq-32b: Embracing the power of reinforcement learning. *URL: https://qwenlm. github. io/blog/qwq-32b*.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *Preprint*, arXiv:2310.11324.

Wenlei Shi and Xing Jin. 2025. Heimdall: test-time scaling on the generative verification. *Preprint*, arXiv:2504.10337.

Yuxia Wang, Minghan Wang, Muhammad Arslan Manzoor, Fei Liu, Georgi Georgiev, Rocktim Jyoti Das, and Preslav Nakov. 2024. Factuality of large language models: A survey. *Preprint*, arXiv:2402.02420.

Qiujie Xie, Qingqiu Li, Zhuohao Yu, Yuejie Zhang, Yue Zhang, and Linyi Yang. 2025. An empirical analysis of uncertainty in large language model evaluations. *Preprint*, arXiv:2502.10709.

Shu-Xun Yang, Cunxiang Wang, Yidong Wang, Xiaotao Gu, Minlie Huang, and Jie Tang. 2025. Stepmathagent: A step-wise agent for evaluating mathematical processes through tree-of-error. *Preprint*, arXiv:2503.10105.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

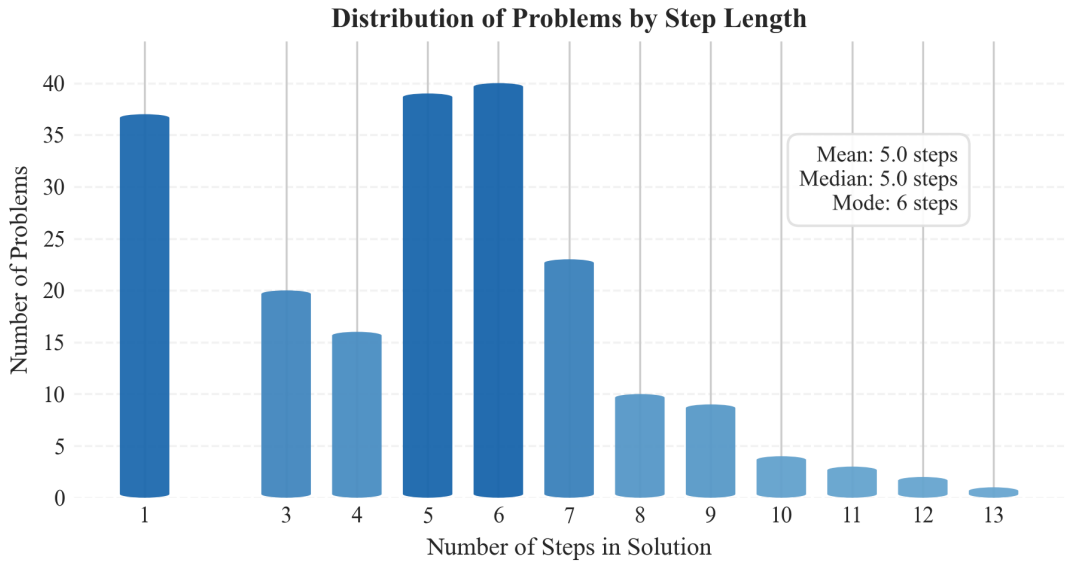

Figure 2: 解题步骤数量的分布。x 轴表示步骤的数量，y 轴表示问题的数量。该分布突显出大部分解答涉及适中的步骤数量，适合进行详细的推理分析。

A 数据集详情

本节详细说明了 MATHOLYMPIADEVAL 数据集的构建和关键特征，这是一个用于评估大型语言模型（LLMs）数学推理的基准。

A.1 数据集组成和问题类型

MATHOLYMPIADEVAL 包含从以下来源获取的 204 个数学问题：中国数学奥林匹克竞赛（CMO，86 个问题）、国际数学奥林匹克竞赛（IMO，62 个问题）和 OpenMathReasoning（56 个问题）。该数据集内容多样，包括 58 个基于证明的问题和 146 个需要最终数值或符号答案的问题。问题难度不一，其中 IMO 的问题代表了最具挑战性的层级。

我们使用基于结果奖励策略训练的开源 LLM 生成了解决方案。提示被设计用来引发完整的、逐步的推理。每个生成的解决方案都经过严格的双重评估过程：为了确保高质量的注释，实施了三轮迭代专家审查过程。推理有缺陷的解决方案根据错误类型进行了分类。如图 3 所示，逻辑谬误（59.8 %）和通过猜测得出的解决方案（34.1 %）是最普遍的错误类别，强调了当前 LLM 推理能力的系统性弱点。

对解的复杂性进行分析（图 2）显示每个问题的平均推理步骤为 5.04，中位数为 5，众数为 6。步骤数的范围从 1 到 13。值得注意的是，约有 75% 的解需要 7 或更少的步骤。这一复杂性水平足够挑战当前模型，但仍然可供人类细致分析推理过程。

A.2 LLM 推理研究的价值

MATHOLYMPIADEVAL 提供了一个关键的基准，其包含了精细的人工注释，专门设计用于推进 LLM 数学推理的研究。其主要贡献在于促进推理过程本身的评估，而不仅仅关注最终答案的正确性。这一区别对于揭示“通过错误推理得到正确答案”的情况至关重要，这种现象可能掩盖模型的真实能力并阻碍进步。通过提供对这些缺陷的见解，该数据集支持开发策略以减轻诸如奖励欺骗等问题，并最终构建更加可靠且可解释的数学推理系统。

A.3 数据集格式

MATHOLYMPIADEVAL 包含 204 条记录。每条记录包括 13 个不同的字段：问题描述、模型生成的解决方案、详细的推理内容、答案正确性的二元评价、人类对推理正确性的评估、参考答案、问题类型（基于证明或寻找答案）、用于生成的 LLM 的详细信息、问题来源、主要错误类型（如果适用）以及推理步骤的总数。

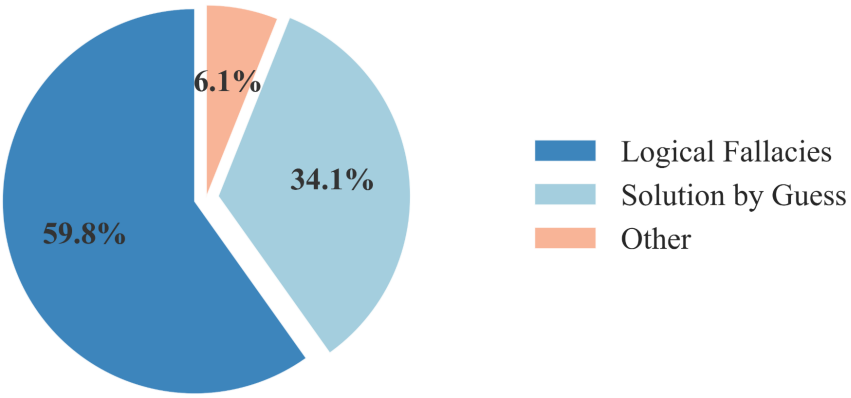

Figure 3: 在大型语言模型生成的解决方案中，由错误推理导致的主要错误类型分布。逻辑谬误和猜测构成了错误的大部分。

B 提示模板

本节介绍了用于数学推理评估的提示模板。模板涵盖对基线的整体评估以及对我们 ParaStepVerifier 的逐步骤验证，并包含错误分类等变体以支持消融研究。

B.1 将 LLM 作为评判者的提示模板

我们的基准方法 LLM-as-a-judge 对整个解决方案的正确性进行一次性评估。

B.2 ParaStepVerifier 提示模板

我们提出的方法，ParaStepVerifier，通过独立评估每个解题步骤，进行细粒度的逐步验证。

C 案例研究

为了说明我们论文的核心发现——模型通常通过存在缺陷的推理来获得正确的最终答案——本节展示了一系列从 MATHOLYMPIADEVAL 数据集中挑选出来的案例研究。这些例子源自 IMO、CMO 和 OpenMathReasoning，展示了各种失败模式，包括“猜测解法”、“计算错误”、“不等式操作错误”和其他“逻辑谬误”。每个案例都是奖励机制漏洞的明显例子，突显了基于结果的正确性与基于过程的稳健性之间的关键差距，并强调了诸如 ParaStepVerifier 之类的更严格评估方法的必要性。

C.1 开放数学推理

Case Study – OpenMathReasoning: Solution by Guess

Problem Description: Find all non-negative integers m and n such that $(2^n - 1) \cdot (3^n - 1) = m^2$.
Error Category: Solution by Guess
Reason for Error: The solution relies on pattern-spotting from a few small examples ($n = 0, 1, 2, \ldots$) and fails to provide a rigorous proof that no solutions exist for larger values of n . Through this flawed and incomplete derivation, it arrives at the correct answer.

Solution:
To find all non-negative integers m and n such that $(2^n - 1)(3^n - 1) = m^2$, we will proceed as follows:
1. **Check small values of n :** - For $n = 0$:

$$(2^0 - 1)(3^0 - 1) = (1 - 1)(1 - 1) = 0 \cdot 0 = 0 = 0^2$$

So, $(m, n) = (0, 0)$ is a solution.

Prompt: LLM-as-a-judge

```
# Instructions

You are a meticulous Mathematical Proof Verifier. Your task is to evaluate the
mathematical correctness of the entire proof for the given problem.

Your evaluation **MUST** strictly follow the structure below, using clear section
headers:

**1. Analysis Process:**
    - Provide a detailed, step-by-step logical verification explaining how the
proof correctly solves the problem.
    - Clearly articulate the reasoning chain, citing the justification for each
key operation (e.g., based on axioms, theorems, definitions).
    - Verify that all claims made in the proof are properly substantiated and that
the final conclusion directly addresses the problem statement.

**2. Final Judgment:**
    - Based *solely* on your findings in Section 1 (Analysis Process), provide a
definitive judgment on the mathematical correctness of the proof.
    - Your judgment **MUST** be precisely one of the following two words:
        - **"CORRECT"**: If the proof is mathematically sound, logically valid,
and properly substantiated.
        - **"WRONG"**: If the proof contains *any* mathematical calculation error
or logical reasoning flaw.

---

# Input

Problem:
{problem}

Proof to evaluate:
{generated_proof}
```

Figure 4: 用于整体解决方案评估的基本 LLM-作为评判者提示。

Prompt: LLM-as-a-judge + Error Classification

```
# Instructions

You are a meticulous Mathematical Proof Verifier. Your task is to evaluate the
mathematical correctness of the entire proof for the given problem.

Your evaluation **MUST** strictly follow the structure below, using clear section headers:

**1. Analysis Process:**
    - Provide a detailed, step-by-step logical verification explaining how the proof
correctly solves the problem.
    - Clearly articulate the reasoning chain, citing the justification for each key
operation (e.g., based on axioms, theorems, definitions).
    - Verify that all claims made in the proof are properly substantiated and that the
final conclusion directly addresses the problem statement.

**2. Error Detection:**
    - Carefully check the proof for **ALL** of the following common types of mathematical
reasoning errors. Analyze each error type:
        *   Solution by Guess (using specific examples instead of general proof/solution)
        *   Trial and Error Approach (unsystematic testing without rigorous solution/proof.
        *   Circular Reasoning (assuming what needs to be proven)
        *   Inequality Manipulation Errors (incorrect application of inequality operations
or bounds)
        *   Logical Fallacies (any flawed reasoning or invalid inference)
    - If any error is found, explicitly state which error(s) occurred and explain *why* it
constitutes an error in this specific proof.

**3. Final Judgment:**
    - Based *solely* on your findings in Section 1 (Analysis Process) and Section 2 (Error
Detection), provide a definitive judgment on the mathematical correctness of the proof.
    - Your judgment **MUST** be precisely one of the following two words:
        - **"CORRECT"**: If the proof is mathematically sound, logically valid, and none
of the errors listed in Section 2 were detected.
        - **"WRONG"**: If the proof contains *any* mathematical calculation error, logical
reasoning flaw, or if any of the errors listed in Section 2 were detected.

---

# Input

Problem:
{problem}

Proof to evaluate:
{generated_proof}
```

Figure 5: 加入错误分类要求的 LLM 作为评判者的提示。

Prompt: ParaStepVerifier

```
# Instructions

You are a meticulous Mathematical Solution Verifier. Your task is to evaluate the
mathematical correctness of the **Current Step**, considering the following context: the
**Problem** description, the **Previous Steps**, and any planned **Future Steps**.

Your evaluation **MUST** strictly follow the structure below, using clear section headers:

**1. Analysis Process:**
    - Provide a detailed, step-by-step logical verification explaining how the **Current
Step** is derived from the **Problem** description and **Previous Steps**.
    - If the **Current Step** cannot be logically derived from the preceding information,
explicitly identify the break in logic and the reason.
    - Referring to **Future Steps** can help understand the overall direction of the
solution, but the evaluation of the **Current Step** must be strictly based on the
information established up to that point.

**2. Error Detection:**
    - Carefully check the **Current Step** for **ALL** of the following common types of
mathematical reasoning errors. Analyze each error type:
        *   Solution by Guess (using specific examples instead of general proof/solution)
        *   Trial and Error Approach (unsystematic testing without rigorous solution/proof.
        *   Circular Reasoning (assuming what needs to be proven)
        *   Inequality Manipulation Errors (incorrect application of inequality operations
or bounds)
        *   Logical Fallacies (any flawed reasoning or invalid inference)
    - If any error is found, explicitly state which error(s) occurred and explain *why* it
constitutes an error in this specific step.

**3. Final Judgment:**
    - Your judgment **MUST** be precisely one of the following two words:
        - **"CORRECT"**: If the proof is mathematically sound, logically valid, and none
of the errors listed in Section 2 were detected.
        - **"WRONG"**: If the proof contains *any* mathematical calculation error, logical
reasoning flaw, or if any of the errors listed in Section 2 were detected.

---

# Input

**Problem:**
{problem}

**Previous Steps:**
{chr(10).join(previous_steps) if previous_steps else "No previous steps"}

**Current Step to Verify:**
{current_step}

**Future Steps:**
{chr(10).join(future_steps) if future_steps else "No future steps"}
```

Figure 6: ParaStepVerifier 使用逐步评估和明确的错误分类的提示。

- For $n = 1$:

$$(2^1 - 1)(3^1 - 1) = (2 - 1)(3 - 1) = 1 \cdot 2 = 2$$

Since 2 is not a perfect square, $n = 1$ is not a solution.
- For $n = 2$:

$$(2^2 - 1)(3^2 - 1) = (4 - 1)(9 - 1) = 3 \cdot 8 = 24$$

Since 24 is not a perfect square, $n = 2$ is not a solution.
- For $n = 3$:

$$(2^3 - 1)(3^3 - 1) = (8 - 1)(27 - 1) = 7 \cdot 26 = 182$$

Since 182 is not a perfect square, $n = 3$ is not a solution.
- For $n = 4$:

$$(2^4 - 1)(3^4 - 1) = (16 - 1)(81 - 1) = 15 \cdot 80 = 1200$$

Since 1200 is not a perfect square, $n = 4$ is not a solution.
- For $n = 5$:

$$(2^5 - 1)(3^5 - 1) = (32 - 1)(243 - 1) = 31 \cdot 242 = 7502$$

Since 7502 is not a perfect square, $n = 5$ is not a solution.
- For $n = 6$:

$$(2^6 - 1)(3^6 - 1) = (64 - 1)(729 - 1) = 63 \cdot 728 = 45864$$

Since 45864 is not a perfect square, $n = 6$ is not a solution.
2. **Analyze the general case:**
Let $A = 2^n - 1$ and $B = 3^n - 1$. Then $A \cdot B = m^2$. For $A \cdot B$ to be a perfect square, A and B must share a common factor d such that d is a perfect square, and the remaining factors of A and B after dividing by d must also be perfect squares.
- Compute the GCD of $2^n - 1$ and $3^n - 1$:

$$\text{GCD}(2^n - 1, 3^n - 1)$$

For $n \geq 1$, $2^n - 1$ is odd and $3^n - 1$ is even. Therefore, the GCD must be odd.
- Check small values of n to see if the GCD is a perfect square: - For $n = 4$:

$$\text{GCD}(2^4 - 1, 3^4 - 1) = \text{GCD}(15, 80) = 5$$

Since 5 is not a perfect square, $n = 4$ is not a solution.
- For $n = 6$:

$$\text{GCD}(2^6 - 1, 3^6 - 1) = \text{GCD}(63, 728) = 7$$

Since 7 is not a perfect square, $n = 6$ is not a solution.
- For $n = 8$:

$$\text{GCD}(2^8 - 1, 3^8 - 1) = \text{GCD}(255, 6560) = 5$$

Since 5 is not a perfect square, $n = 8$ is not a solution.
- For $n = 10$:

$$\text{GCD}(2^{10} - 1, 3^{10} - 1) = \text{GCD}(1023, 59048) = 11$$

Since 11 is not a perfect square, $n = 10$ is not a solution.
3. **Conclusion:**
From the above analysis, the only value of n that results in a perfect square is $n = 0$. Therefore, the only non-negative integer solution is:

$$\boxed{(m, n) = (0, 0)}$$

我们使用人工标注者进行评估。标注者来自研究生院，并由资助支持。所有标注者都同意共享数据。数据收集协议已获得伦理审查委员会的批准。