CoDiQ：面向可控困难问题生成的测试时扩展

Zhongyuan Peng[1,3], **Caijun Xu**[1,2], **Changyi Xiao**[1], **Shibo Hong**[1], **Eli Zhang**[†3], **Stephen Huang**[3], **Yixin Cao**[†1,2]

[1]复旦大学， [2]上海创新研究院， [3]M-A-P

[†]通讯作者

arXiv:2602.01660v1 [cs.CL] 2 Feb 2026

摘要

大型推理模型（LRM）通过在具有挑战性的竞赛级问题上训练，能够显著受益。然而，现有的自动化问题合成方法缺乏精确的难度控制，计算成本高昂，而且难以大规模生成竞赛级问题。本文提出了**CoDiQ** （**Co**可控**Di**难度**Q**问题生成），这是一种新颖的框架，可通过测试时扩展实现细粒度难度控制，同时确保问题可解性。具体而言，我们首先发现了一种测试时扩展趋势（增加推理词元预算会提高难度，但会降低可解性），并确定了定义模型生成有效高难度问题能力上限的内在属性。随后，我们基于 Qwen3-8B 开发了**CoDiQ-Generator**，提升了难题生成的能力上限，使其尤其适用于构造具有挑战性的问题。在 CoDiQ 框架的基础上，我们构建了**CoDiQ-Corpus** （包含 44K 个竞赛级问题序列）。人工评估表明，这些问题比 LiveCodeBench/AIME 上的问题更具挑战性，且可解率超过 82%。在 CoDiQ-Corpus 上训练大型推理模型，能够显著提升其推理性能，验证了使用难度受控的训练问题进行扩展可以增强推理能力。我们开源了 CoDiQ-Corpus、CoDiQ-Generator 及相关实现，以支持相关研究。

日期： 2026年2月2日**通讯作者：** yxcao@fudan.edu.cn, zhangge.eli@bytedance.com**项目页面：** https://github.com/ALEX-nlp/CoDiQ

1 引言

大型推理模型（LRMs）的快速发展展现了其在复杂推理方面的非凡能力，近期研究在数学和编程领域的各项具有挑战性的基准测试中取得了令人瞩目的表现 [22, 23, 38]。推动这些进步的一个关键因素，是拥有真正能够充分考验推理能力的高质量训练和评估数据；然而，这类数据十分稀缺。重要的是，与科学发现非常相似，找到合适的难题可能与解决难题同等关键。随着难度不断提高，可靠的问题构造需要专家知识和谨慎验证，这使得完全由人工驱动的流水线成本高昂且难以扩展。

本文旨在扩大高难度题目的合成规模，同时确保题目定义良好且可解。近期研究探索了数学和编程领域的多种方法，从人在回路的方法 [26] 和对抗式生成 [33]到迭代进化策略 [9,

22, 38]。

然而，大规模提升难度面临三大挑战。首先，存在生成器能力上限：模型通常难以生成远超自身可靠推理能力的题目，导致进展停滞。其次，可解性—复杂度权衡 意味着一味强求复杂度往往会破坏逻辑一致性，产生看似“困难”却无法求解或定义不清的题目。最后是难度定义与控制。 由于难度既无法直接观测，也没有统一标准，若没有可测量替代指标，“提高难度”便无法控制，从而使课程式训练变得脆弱。

为应对这些挑战，我们提出了*CoDiQ*（可控困难问题生成），这是一个将测试时扩展引入问题生成的框架，并通过三项关键创新系统地提升难度。首先，我们设计了六种难度增强策略，并通过强化学习训练 **CoDiQ-Generator**，以合成超越零样本基线的问题。其次，我们开发了*CoDiQ* 流水线，这是一个采用混合验证的迭代框架，可在提升复杂度的同时确保逻辑一致性。第三，我们通过基于大语言模型的排序和价值网络建立了相对难度范式，利用连续分数量化难度，从而精确划分等级。基于 CoDiQ，我们构建了 **CoDiQ-Corpus**，其中包含 4.4 万条竞赛级数学与编程问题序列。人工评估和实验表明，我们的方法能够生成高质量数据，并显著提升下游推理性能。

我们的主要贡献如下：

- **难度增强策略**。我们提出六种系统化策略，指导大语言模型在问题生成中注入困难要素，从而合成难度高于零样本生成基线的问题。

- **难度的测试时扩展趋势**。我们发现了一种将测试时计算量与问题难度联系起来的扩展趋势，刻画了模型生成有效高难度问题的能力上限。

- **CoDiQ-Corpus**。我们基于 CoDiQ-Generator 构建了一个包含 4.4 万道竞赛级数学和编程问题的数据集。实验表明，与现有基线相比，在 CoDiQ-Corpus 上训练能显著增强大型推理模型的推理能力。

我们将开源 CoDiQ-Corpus、CoDiQ-Generator 以及全部实现，以支持未来研究。

2 相关工作

生成既困难又有效的问题日益被视为扩展推理进展的关键杠杆：它将训练分布扩展到稀缺的人工策划问题之外，持续提供前沿级监督，并提供一种在难度不断提升的条件下进行泛化测试的可控方式 [12, 21, 29]。因此，近期研究投入了大量精力来合成兼具智力挑战与形式正确性保证的竞赛级问题。

基于提示和智能体的问题合成流水线。 一种主流范式将难题创建视为由提示驱动或由智能体执行的工作流程：系统从种子问题、概念或人工编写的解题结构出发，通过自我批评、筛选和验证信号迭代优化候选项，以确保其适定性 [20, 31, 39]。在数学领域，PromptCoT [38] 通过概念采样和结构化设计线索驱动生成，明确引导模型产生类似专家的问题构造推理，然后采用拒绝采样保留连贯且难度较高的实例。CogAtom [5] 则将人工解答分解为可复用的认知原子，构建原子图，并通过约束重组来合成新问题，从而系统地探索组合式设计空间。对于编程任务，可靠性尤为重要：一个有效实例不仅需要问题陈述，还需要精确的输入输出规范、有意义的约束，以及反捷径测试套件。AutoCode [40]展示了一种闭环出题流水线，它联合生成问题陈述和参考解答，并通过自动化测试生成与交叉验证筛除规范不充分或适定性不佳的任务。总体而言，这些方法虽然有效，但往往依赖复杂的多步协调和大量事后筛选来维持有效性。

训练生成器以生成难题。另一类研究侧重于训练专用生成器，以摊销多步智能体流程的成本，从而以较低的边际成本实现大规模困难问题合成 [7, 15, 34]。例如，ScaleQuest [10] 通过问题微调和问题偏好优化来提升问题生成能力，使生成结果更符合可解性和难度要求。ScaleDiff [24] 首先高效识别困难实例，再针对困难子集训练专用生成器，以拓展高难度分布的上尾。总体而言，生成器训练方法具有良好的扩展性，但仍存在一些常见局限：难度控制通常较为粗糙，而有效性仍在很大程度上依赖事后筛选或人工验证。

与以往的合成流程和生成器训练方法不同，我们的方法以测试时扩展为核心机制，在可验证的可解性下实现细粒度难度控制：我们在推理时显式扩展实例难度，同时通过自动验证强制保证正确性，而不是依赖筛选。这使得我们能够对困难但可解的问题进行系统性的前沿跟踪，同时在规模化条件下保持有效性约束完整且可控。

3 方法

3.1 概述

我们的方法旨在通过使大型推理模型能够合成难度逐步提升且有效的问题，赋予其可扩展的测试时问题生成能力。为此，我们首先引入六种难度增强策略（§3.2），明确引导大型推理模型进行难度扩展推理和困难问题构造。这些策略被融入 CoDiQ 流水线（§3.3）中，该流水线整合了两个验证模块——难度估计（§3.3.1）和可解性验证（§3.3.2）——以共同调节难度与有效性。借助该流水线，我们构建了 CoDiQ-Bench，在统一评估框架下系统地对模型的问题生成性能进行基准评测。随后，我们通过强化学习（§3.5），利用源自流水线的反馈信号进一步提升所合成问题的难度与可靠性，开发了专门的 CoDiQ-Generator。最后，我们基于 CoDiQ-Generator 构建了 CoDiQ-Corpus，这是一个包含 44K 道竞赛级数学和编程问题的数据集。详细统计数据见附录 E，其分布如图 1所示。

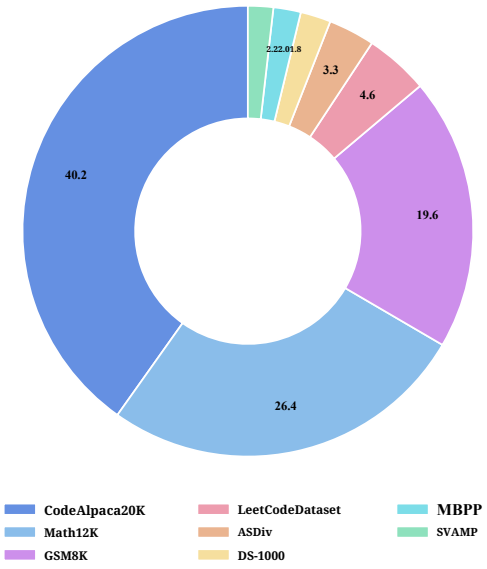

图 1 CoDiQ-Corpus 数据集的分布

3.2 难度增强策略

为了系统性地提升问题难度，超越简单的提示（例如“让它更难”），我们设计了六种难度增强策略（详见附录 L），作为大语言模型明确的认知支架。这些策略——维度与约束、数学抽象、逆向与构造、状态爆炸、定理伪装和边界情况与严谨性工程——引导模型系统性地注入算法难度，确保难度源于推理深度，而非表面上的修改。

3.3 CoDiQ 流水线

基于难度注入策略（§3.2），我们提出了 CoDiQ 流水线（算法 1），通过迭代细化系统地提升难度。该流水线实现了一个演化循环，其中种子问题 Q_0 逐步演化为更困难的变体 $\{Q_1, \ldots, Q_n\}$，最多经过 $T_{\max} = 8$ 轮次。在每次迭代中，模型都会收到“你能让它变得更困难吗？”这一提示，以触发更深层次的推理。

为确保生成质量，该流水线整合了两个核心验证模块： 难度估计（§3.3.1）和 可解性验证 （§3.3.2）。该过程依据严格的停止规则（§3.3.3）终止。

3.3.1 难度估计

为严格实施单调难度约束，我们需要一种稳健机制来检测难度回退。由于 CoDiQ 面向模型能力的前沿，标准绝对评分会受到饱和效应的影响——模型对高难度查询一律给出很高的分数——从而使直接比较失效。因此，我们采用相对难度范式，包含两种互补方法：显式*LLM* 排序（§3.3.1），以辨别相对难度，以及隐式价值网络评分（§3.3.1），以捕捉内部不确定性。最后，我们对这些离散排序（§3.3.1）进行归一化，以消除粒度偏差。

算法 1 CoDiQ 迭代流程

输入： 种子问题 Q_0，最大轮数 T_{max}
输出： 演化问题 $\mathcal{Q}$
初始化 $\mathcal{Q} \leftarrow \emptyset$, $d_0 \leftarrow$难度(Q_0)
对于 $i = 1$ **到** T_{max} **do**
  $Q_i \leftarrow$大语言模型(Q_{i-1})
  $d_i \leftarrow$难度(Q_i)
  如果 有效$(Q_i) =$假 **或** $d_i < d_{i-1}$ **则**
    中断
  结束如果
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q_i\}$
结束循环
返回 $\mathcal{Q}$

LLM 排序。 为自适应分配推理预算，我们使用 Doubao-Seed-1.8 [25] 进行列表式难度估计。给定一批查询 $\mathcal{Q} = \{q_1, \ldots, q_n\}$，模型按照附录 I 中的提示模板，根据感知难度对它们进行排序。为缓解位置偏差，我们应用 随机洗牌 τ（$\mathcal{Q}$）于排序之前。模型输出结构化 JSON 结果，我们从中提取排序索引以映射计算预算 K，从而为更难的查询分配更多样本。

价值网络评分。 为高效估计问题难度，我们通过分析模型的推理轨迹，扩展了 [41] 所采用的基于隐藏表示的方法。我们使用 QWEN3-8B 在最多 4,096 个 Token 的采样窗口内提取隐藏状态。为捕捉推理的关键早期阶段，我们采用二次采样策略（公式 §4），使生成起始阶段具有更高的采样密度。随后，将这些表示输入通过二元交叉熵训练的轻量级多层感知机，以预测正确概率 $y \in \{0, 1\}$，其训练数据涵盖标准基准测试和竞赛级基准测试 [8, 16, 17, 19]。该方法展现出较强的问题难度区分能力（见附录C.5）。在推理阶段，预测概率可作为 LLM 感知难度的代理指标，分数越低表示难度越高。详细实现见附录C。

难度归一化。 为了将 §3.3.1 和 §3.3.1 中的离散分组排名转换为连续分数，我们采用线性缩放。给定 G 个按难度从低到高排序的难度组 $\mathcal{G} = \{g_1, \ldots, g_G\}$，组 g_j 中问题 q_i 的归一化难度为：

$$d_i = \frac{j-1}{G-1}, \quad j \in \{1, \ldots, G\}. \tag{1}$$

这将离散的排 名映射到 $[0, 1]$, ，其中 d_i 用作自适应计算分配的缩放因子 n.

3.3.2 可解性验证

虽然难度估计确保复杂度单调增长，但并不能保证逻辑有效性。为避免生成无效或无法求解的实例，我们使用通义千问3-32B [35]验证生成实例的可解性。按照附录J中的模板，模型以JSON 格式生成响应，我们从中提取可解性状态和置信度分数。只有经验证可解且置信度高的实例才会被保留。

3.3.3 终止准则

为维护问题轨迹的完整性，流水线执行严格的停止规则。出现以下情况时，迭代过程立即终止：（1）**非单调难度**，即生成的问题 Q_i 难度分数低于其前序问题；或（2）**不可解性**，即候选项 Q_i 被标记为无效。若在步骤 i终止，则丢弃无效候选项，并由流水线输出序列$\{Q_1, \ldots, Q_{i-1}\}$。案例研究见附录A，失败类型分析见附录B。

3.4 CoDiQ-Bench

表 1 CoDiQ-Bench 数据集统计信息。

统计信息	数量
#问题	200
- 数学	100
- 代码	100
问题词元长度	
- 最大值/最小值/平均值	726/9/128。2

为系统评估大型推理模型的问题生成能力，我们首先构建了 CoDiQ-Bench，这是一个精心整理的数据集，包含编程和数学领域的 200 个精选案例（表§1）。对于编程任务，我们分别从 CodeAlpaca_20K（通用编程）和 LeetCodeDataset（算法挑战）中随机抽取 50 个案例。对于数学推理，我们分别从 GSM8K（小学题目）和 MATH12K（数学问题求解）中抽取 50 个案例。我们有意聚焦于相对简单的问题，以建立一个基线基准；关于可解性和质量的详细筛选标准见附录§D。

3.5 CoDiQ-Generator

为进一步提升 CoDiQ 流水线生成高难度、高质量问题的能力，我们通过强化学习开发了 CoDiQ-Generator。通过针对性地优化模型的出题行为，并提供定向奖励信号，我们旨在提升合成问题的有效性和难度缩放能力。

3.5.1 强化学习数据构造

我们通过捕捉通义千问3-8B 在 CoDiQ 流水线（第 $\mathcal{D}_{RL}$ 节）中的关键失效模式，构建强化学习数据集 $\mathcal{D}_{RL}$ 3.3。我们并不追求最大化绝对难度，而是针对模型特定的 **能力边界** [37]，识别模型在第 1轮至 $i-1$轮成功生成有效问题、但在第 i轮失败（例如因不可解性或难度停滞）的演化轨迹。我们收集这些边界实例来构建训练对，从而将模型的“临界点”转化为精确的学习信号。

为确保广泛覆盖不同领域，我们使用来自多个成熟基准的种子问题（Q_0）初始化流水线。数学领域的样本取自 Math12k [14], GSM8K [8], SVAMP [6],和 ASDiv [36]。代码生成方面，我们使用 Code Alpaca [4], LeetCodeDataset [32], MBPP [2], 和 DS-1000 [18]。根据上述特定边界条件进行筛选后，最终数据集 $\mathcal{D}_{RL}$ 包含**1,173**个高质量样本。

3.5.2 强化学习训练范式

强化学习优化（*RL*）。 近期 R1 式方法的成功证明了使用离散的基于规则的奖励进行在线强化学习的有效性 [27]。在我们的流水线中，Qwen3-8B [35] 进一步通过基于可解性置信度、难度递进和

问题有效性检查所产生的信号进行优化。基于 §3.5.1 中描述的数据集，我们采用基于规则的强化学习方法来优化模型的判断推理能力。具体而言，我们在 [27] VeRL 强化学习框架 [28]中使用 GRPO 算法。

为确保优化过程平稳，我们设计了一种感知难度的奖励函数，在保证有效性的同时实现渐进式难度缩放。给定置信度下界

$$\text{conf} = \max(0.5, \text{confidence}(x))$$

以及难度变化

$$\Delta(D) = d_i - d_{i-1} \in [-1, 1] \tag{2}$$

对于迭代 $i \in \{1, \ldots, R\}$，其中 d_i 通过公式（1）计算，且 R 表示最大进化轮次数:

$$r = \begin{cases} 0, & \text{if invalid} \\ 0.6 \cdot \text{conf}, & \text{if } \Delta(D) = 0 \\ 0.2 \cdot \text{conf} + 0.8 \cdot (0.8 + 0.2 \cdot \Delta(D)), & \text{if } \Delta(D) > 0 \end{cases} \tag{3}$$

其中，无效情况包括无法求解的问题、重复输出或负向难度变化（$\Delta(D) < 0$）。

4 实验

4.1 实验设置

基线为评估 CoDiQ 提示和 CoDiQ-Generator 的有效性，我们将其与具备内在测试时扩展能力、支持扩展推理的模型进行比较。这些基线模型包括旗舰模型（GLM-4。6 [11]）和参数规模较小的模型（GPT-OSS-20B [1], GLM-Z1-9B-0414 [11], 以及 Qwen3 系列 [35]: Qwen3-0.4B、Qwen3-1.7B、Qwen3-4B、Qwen3-8B、Qwen3-14B、Qwen3-32B）。所有这些模型都使用第 3.3 节所述的 CoDiQ 流水线进行生成。

评估指标。我们采用两项指标来量化问题难度：（1）**DS-LLM**：由 Doubao-Seed-1. 8 [25] 模型估算的难度分数（详情见第 3.3.1 节）。（2）**DS-VN**：根据价值网络（VN）得出的难度分数（详情见第3.3.1 节）。两项分数均归一化至范围 [0, 1]，并以百分比（0%-100%）报告，数值越高表示难度越大。所有报告的分数均为 CoDiQ-Bench 中各问题的平均值。

4.2 主要结果

4.2.1 最大可解难度

为了评估大型推理模型（LRMs）在我们提出的框架中的问题生成能力，并确定后续难题合成的最优生成器，我们进行了对比分析。具体而言，我们将不同的大型推理模型实例化为 CoDiQ 流水线中的骨干生成器，并在 CoDiQ-Bench 上评估它们所生成问题的难度。

CoDiQ 提示的有效性。我们首先评估 CoDiQ 提示在引导深度推理以合成难度方面的效果。如表 2所示，应用 CoDiQ 提示后，所有受评架构的推理词元使用量都大幅增加。测试时计算量的显著增长表明，该提示成功触发了更长的推理轨迹，使模型能够构建更复杂的约束和逻辑。因此，在我们的提示引导下，大多数基线模型生成的问题难度都有明显提升。

表 2 不同长链式思维模型在 CoDiQ-Bench 上的表现。基于 CoDiQ-Bench 上 8 轮中生成的、难度未发生退化的可解问题的最高难度进行组别排名。每项指标的最佳、次佳和第三佳分数分别以 $\boxed{box}$ 、**粗体**和下划线标示。

模型	轮次	Token	DR-LLM	DR-VN	DR（平均值）
直接提示					
GPT-OSS-20B	2.9	5528.2	68.5	**74.4**	**71.5**
GLM-4.6	2.8	3385.8	**71.2**	65.8	68.5
Qwen3-32B	2.3	1239.3	50.6	54.8	52.7
Qwen3-8B	3.4	1130.5	39.2	59.6	49.4
GLM-Z1-9B-0414	2.7	1229.8	48.8	43.7	46.3
Qwen3-14B	3.1	2076.4	45.9	44.4	45.2
Qwen3-4B	4.2	1419.7	36.8	40.4	38.6
Qwen3-1.7B	3.3	844.5	25.6	37.1	31.4
Qwen3-0.6B	2.4	314.3	17.2	35.0	26.1
CoDiQ 提示（我们的方法）					
GLM-4.6	2.7	7143.8	73.2	83.3	78.3
GPT-OSS-20B	2.1	8057.3	63.8	61.5	62.7
Qwen3-32B	2.2	4893.6	63.0	46.5	54.8
Qwen3-14B	2.6	5281.9	53.9	44.2	49.1
Qwen3-4B	2.8	4422.3	49.1	42.7	45.9
Qwen3-8B	2.4	4155.6	49.8	41.9	45.8
GLM-Z1-9B-0414	1.7	3638.3	54.7	30.0	42.4
Qwen3-1.7B	1.4	2975.7	32.3	37.3	34.8
Qwen3-0.6B	1.0	2052.7	22.4	29.2	25.8
CoDiQ 生成器（我们的方法）					
CoDiQ-Gen-8B	**3.4**	**7499.6**	58.9	58.1	58.5

CoDiQ-Generator 的优越性。值得注意的是，在生成高复杂度实例方面，我们的 CoDiQ-Gen-8B 优于规模大得多的通义千问3- 32B。我们将这一性能提升归因于第 3.5.1 节所述的强化学习对齐。通过针对可解性和难度递进进行优化，强化学习训练使 CoDiQ-Generator 在迭代演进过程中保持较高的有效率。这种稳定性使模型能够让生成流程持续更多轮次——超过基线模型的迭代深度——从而使复杂度单调递增，避免因不可解性而过早终止。

4.2.2 难度指标比较

为进一步验证大型推理模型消耗的令牌数能否估计问题难度，我们强调图 2 所示令牌量与难度排名之间的正相关性。我们进一步分析令牌消耗与既定指标（DR-LLM 和 DR-VN）之间的相关性来验证这一关系，得到的皮尔逊系数分别为 $r = 0.8299$ （$p < 0.001$）和 $r = 0.8545$ （$p < 0.001$）。这些结果证实，只要问题复杂度处于评估器的能力范围内，并采用一致的扩展方法，计算成本就可以作为难度的可靠代理指标。

4.3 消融研究

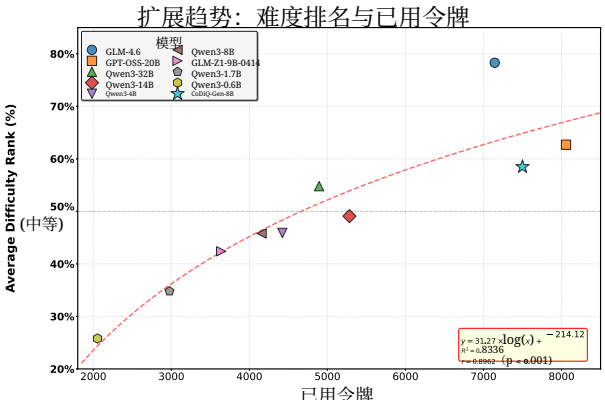

图 2 CoDiQ-Bench 上的问题难度扩展。散点图展示了使用 CoDiQ 提示的模型的平均推理 Token 数与难度排名（DR-AVG）之间的关系。每个点代表一个模型，体现了推理计算量增加与生成问题难度提升之间的正相关关系。

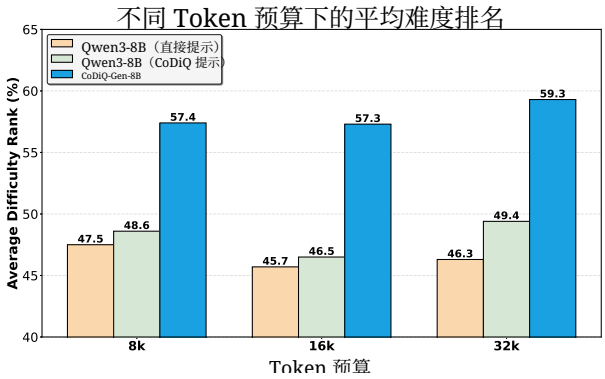

图 3 不同 Token 预算下 CoDiQ-Bench 上的模型性能。三个模型变体（使用直接提示的 Qwen3-8B、使用 CoDiQ 提示的 Qwen3-8B，以及 CoDiQ-Gen-8B）在不同 Token 预算限制（8k、16k、32k）下的平均难度排名（%）。分数越高，表示模型处理难题的性能越好。

表 3 不同长链式思维模型在 CoDiQ-Bench 上的性能。 组别排名基于在 CoDiQ-Bench 上经过 8 轮生成的问题的最高难度。各指标的最佳、次佳和第三佳分数分别以 *box* 、**粗体** 和下划线标示。

模型	轮次	Token	DR-LLM	DR-VN	DR（平均值）
直接提示					
Qwen3-8B	6.0	2439.7	33.5	39.1	36.3
Qwen3-14B	5.6	4927.4	45.6	55.6	50.6
Qwen3-32B	5.7	4124.9	**65.3**	47.5	56.4
CoDiQ 提示（我们的方法）					
Qwen3-8B	5.8	7282.2	53.5	53.3	53.4
Qwen3-14B	5.6	9590.2	58.6	63.1	60.9
Qwen3-32B	5.7	**9762.4**	74.6	**65.0**	69.8
CoDiQ 生成器（我们的方法）					
CoDiQ-Gen-8B	**5.9**	12591.6	52.6	72.2	**62.4**

4.3.1 难度生成的上限

在第 4.2.1 节中，我们在保持解答有效性的约束下评估了最大可解难度。然而，这一可解性要求本身限制了难度上限，因为高度复杂的问题可能并非本质上无法求解，而是超出了当前模型生成有效解答的能力。为探索独立于解答生成约束的难度合成理论上界，我们通过从 CoDiQ 流水线中移除可解性验证模块来开展消融研究。

结果表明，与标准提示相比，融入 CoDiQ 提示显著提高了各骨干模型的难度上限。值得注意的是，尽管参数更少，我们的 CoDiQ-Gen-8B 生成问题的难度上限仍超过了 Qwen3-14B。这表明，我们的专门调优和提示策略有效释放了模型合成高难度

复杂逻辑结构的潜力，即使是在较小的架构中也是如此。

4.3.2 最大令牌预算的影响

我们进一步考察难度缩放相对于计算成本的效率。图 3 展示了 CoDiQ 流水线在严格限制累计令牌使用量的情况下生成的可解 问题的最大难度。为模拟资源受限环境，我们施加严格的累计令牌预算，涵盖生成和验证阶段。如果某一轮迭代中的令牌总消耗超过阈值，则舍弃该轮，并报告此前轮次中难度最高的有效问题。比较分析表明，在所有令牌预算阈值下，CoDiQ-Gen-8B 均表现出明显优势，其生成的问题难度分数始终高于基线模型。此外，我们发现，采用 CoDiQ 提示的 Qwen3-8B 性能优于使用直接提示的版本。这一性能差距验证了 CoDiQ方法能够有效利用计算资源，在保持问题可解性的同时最大限度地提高问题难度。

4.4 扩展趋势分析

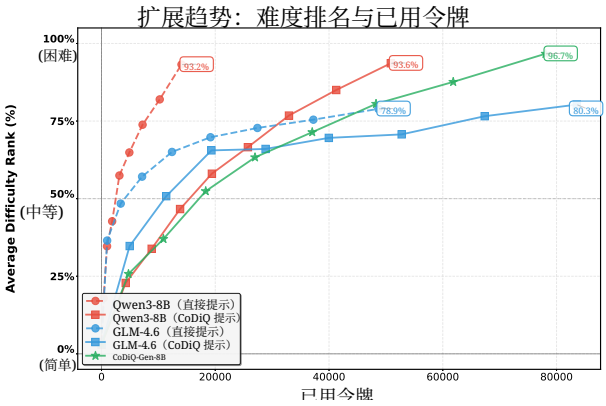

图4 CoDiQ-Bench 上的问题难度缩放。不同长链式思维模型在 8 轮中生成问题的归一化平均难度排名。排名越高，表示问题难度越高，模型性能越好。

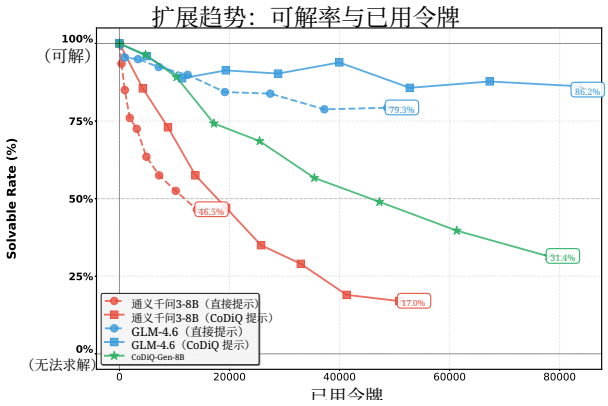

图5 CoDiQ-Bench 上的问题可解性扩展。 不同长链式思维模型在 8 轮中生成问题的可解率。数值越高表示问题质量越好。

前述分析确定了不同大型推理模型的性能上限，同时识别了它们的最大可解难度（§4.2.1）及其理论上界（§ 4.3.1）。然而，这些指标代表的是静态终点。为了理解这些模型如何达到如此复杂度，我们现在转向对生成动态的细粒度分析。在本节中，我们追踪特定模型组内难度与可解性相对于推理计算的缩放趋势（更多结果见附录 F）。

4.4.1 难度缩放

图 4 展示了 8 轮生成过程中问题复杂度的演变。与直接提示相比，*CoDiQ* 提示显著激发了更深层次的推理，导致令牌消耗大幅增加。尽管大多数模型的难度总体呈持续上升趋势，但大参数模型往往会在后期轮次趋于饱和。我们认为，这种平台期是由于令牌消耗量较大，可能已接近模型生成能力或难度评估器的上限。此外，从单模型视角进行的这项分析也印证了第4.2.2 节中的发现，进一步强化了这一结论：令牌量是难度的可靠指标。

4.4.2 可解性扩展

我们考察了可解率如何随难度增加而下降（图5）。这种下降揭示了问题难度与有效性之间的根本性权衡。由此得出三项重要发现：

- **最先进模型的稳健性：**旗舰模型（例如 GLM-4.6）在所有难度等级上均保持较高的可解性，展现出均衡的生成与验证能力。
- **过度推理陷阱：**较小的模型在 CoDiQ 下会出现有效性崩溃，因为它们生成的复杂度超出了自身的推理能力。
- **强化学习对齐的有效性：**CoDiQ-Gen-8B 通过强化学习打破了这一退化模式，成功将难度缩放与有效性损失解耦。

4.5 CoDiQ-Corpus 的有效性

为全面评估该语料库的价值，我们开展了多维评估，重点关注难度（第4.5.1节）、质量（第4.5.2节）以及训练有效性（第4.5.3节）。我们首先表明，CoDiQ-Corpus 在题目难度上显著超越现有竞赛级基准。随后，我们通过严格的人工评估，验证所生成题目的逻辑合理性与可解性。最后，我们在课程学习框架中使用该语料库，验证其实际效用，证明其能够推动推理模型持续改进。

4.5.1 难度比较

为验证 CoDiQ-Corpus 的难度更高，我们从每个数据集中随机抽取 300 道题目，包括 CoDiQ-Corpus、AIME [30], NuminaMath-1。5 [17], LiveCodeBench [16], 和 Code-Contests [19]„ 并采用第 4.1 节中的排名方法对它们进行比较。如表 4 所示，与现有竞赛级数据集相比，我们的 CoDiQ-Corpus 难度显著更高。

表4 数据集难度比较。每项指标的最佳、次佳和第三佳分数分别以 $\boxed{box}$、**粗体**和下划线标示。

数据集	DR-LLM	DR-VN	DR（平均值）
基线			
AIME（1983—2024）	**57.9**	<u>45.1</u>	**51.5**
NuminaMath-1.5	27.5	32.0	29.8
LiveCodeBench	39.4	**45.2**	42.3
Code-Contests	<u>47.2</u>	41.0	44.1
CoDiQ 数据集（本文）			
CoDiQ-Corpus	$\boxed{91.4}$	$\boxed{82.8}$	$\boxed{87.1}$

表 5 模型性能比较。各项指标中，最佳、次佳和第三佳的分数分别以 $\boxed{box}$、**粗体**和下划线标示。

模型	**MATH-500**	**AIME 2024**
基线		
Qwen3-4B	94.4	63.1
Qwen3-RL-4B	<u>95.2</u>	64.3
课程学习模型（我们的方法）		
CoDiQ-L1-4B	**96.0**	<u>65.0</u>
CoDiQ-L2-4B	94.8	**66.7**
CoDiQ-L3-4B	$\boxed{96.0}$	$\boxed{70.6}$

4.5.2 人工质量评估

为验证 CoDiQ-Corpus 和 CoDiQ 流水线的可靠性，我们对$N = 200$ 从已接受的CoDiQ语料库和拒绝案例中抽取的分层样本进行了人工评估。三位博士专家分别评估了清晰度、完整性和推理有效性（附录G），评估结果显示出较高的一致性（Fleiss'$\kappa = 0.76$）。

结果显示，被接受实例的精确率为 82%，拒绝案例的阴性预测值为 90%。值得注意的是，对假阴性（被错误拒绝的有效问题）进行错误分析，经验性地揭示了验证器悖论：这些实例在逻辑上是成立的，但超出了验证器的推理范围，导致模型将其误判为“无法求解”，而非“困难”。这证实了当前流程上限受验证器能力制约。

4.5.3 训练有效性验证

通过课程学习进行强化学习验证。CoDiQ 的一项显著优势在于其内在的可控性。通过调整 Token 预算，它能够生成难度逐步递增的问题序列，自然而然地促进课程学习策略 [3]，使其与模型不断发展的能力相契合。

基于此，我们采用多阶段强化学习范式，通过依次训练模型 CoDiQ-L$_i$-4B（$i \in \{1, 2, 3\}$），每个阶段 i 均使用难度逐渐递增的数据集子集。奖励由通义千问3-32B通过评估回答质量并进行加权汇总得出（详情见附录 H）。我们将该方法与原版 Qwen-4B 以及 Qwen3-RL-4B 进行比较；后者是在未经难度分层的原始数据集上通过标准强化学习训练的基线模型。在 MATH-500 和 AIME 2024 上的评估结果（表5）表明，与标准训练范式相比，我们的预算控制课程框架显著提升了性能，从而验证了 CoDiQ-Corpus 的有效性与实用价值。

5 结论与局限性

我们提出了 *CoDiQ*，这是一个原则明确的框架，可大规模合成可验证的高难度推理问题。通过借助测试时扩展突破生成器能力上限，并利用混合验证流程减少“虚假困难”样本，我们成功地通过强化学习训练了 **CoDiQ-Generator** 。由此得到的 **CoDiQ-Corpus** 采用预算驱动的难度分层，其在课程学习中的有效应用验证了该方法的优越性。我们将流水线开源，以促进未来对扩展定律和自动化课程学习的研究。

然而，我们承认自身存在一些局限。目前，我们的范围仅限于英语数学/代码任务，而验证成本限制了实时使用。最关键的是，我们的流水线面临着验证器悖论：依赖固定容量的验证器会造成认知上限，使得那些超出验证器能力但确实有效的问题面临被判定为无法求解并丢弃的风险。未来的工作必须应对这一可扩展监督挑战。

影响声明

我们的工作为提升合成推理数据的难度，同时保持逻辑有效性，提供了一个基础框架。通过将问题复杂度与人工筛选解耦，这项研究有助于推动人工智能系统在数学和编程领域发展出更强健的推理能力。尽管这能加快模型性能的提升，但也凸显了纳入严格可解性约束的重要性，以防止自动化训练循环中的数据质量下降。

参考文献

[1] Sandhini Agarwal、Lama Ahmad、Jason Ai、Sam Altman、Andy Applebaum、Edwin Arbus、Rahul K Arora、Yu Bai、Bowen Baker、Haiming Bao 等。gpt-oss-120b 与 gpt-oss-20b 模型卡。arXiv 预印本 arXiv:2508.10925，2025。[2] Jacob Austin、Augustus Odena、Maxwell Nye、Maarten Bosma、Henryk Michalewski、David Dohan、Ellen Jiang、Carrie Cai、Michael Terry、Quoc Le 等。利用大语言模型进行程序合成。arXiv 预印本 arXiv:2108.07732，2021。[3] Yoshua Bengio、Jérôme Louradour、Ronan Collobert 和 Jason Weston。课程学习。载于第 26 届年度国际机器学习会议论文集，第 41–48 页，2009。[4] Sahil Chaudhary。Code Alpaca：用于代码生成的指令遵循型 Llama 模型。https://github.com/ sahil280114/codealpaca，2023。[5] Zhuofan Chen、Jiyuan He、Yichi Zhang、Xing Hu、Haoxing Wen、Jun Bai 和 Wenge Rong。Cogatom：从认知原子到大语言模型的奥林匹克级数学推理，2025。URL https://arxiv. org/abs/2509.17318。[6] ChilleD。SVAMP 数据集，2024。[7] Bryan R Christ、Jonathan Kropko 和 Thomas Hartvigsen。MathWell：利用教师标注生成教育数学文字题，2024。URL https://arxiv.org/abs/2402.15861。[8] Karl Cobbe、Vineet Kosaraju、Mohammad Bavarian、Mark Chen、Heewoo Jun、Lukasz Kaiser、Matthias Plappert、Jerry Tworek、Jacob Hilton、Reiichiro Nakano、Christopher Hesse 和 John Schulman。训练验证器以解答数学文字题。arXiv 预印本 arXiv:2110.14168，2021。[9] Yuyang Ding、Xinyu Shi、Juntao Li、Qiaoming Zhu、Min Zhang 等。通过从头开始进行可扩展的问题合成，释放大语言模型的推理能力。arXiv 预印本 arXiv:2410.18693，2024。[10] Yuyang Ding、Xinyu Shi、Xiaobo Liang、Juntao Li、Zhaopeng Tu、Qiaoming Zhu 和 Min Zhang。通过从头开始进行可扩展的问题合成，释放大语言模型的推理能力，2025。URL https://arxiv.org/abs/2410.18693。[11] Team GLM、Aohan Zeng、Bin Xu、Bowen Wang、Chenhui Zhang、Da Yin、Diego Rojas、Guanyu Feng、Hanlin Zhao、Hanyu Lai、Hao Yu、Hongning Wang、Jiadai Sun、Jiajie Zhang、Jiale Cheng、Jiayi Gui、Jie Tang、Jing Zhang、Juanzi Li、Lei Zhao、Lindong Wu、Lucen Zhong、Mingdao Liu、Minlie Huang、Peng Zhang、Qinkai Zheng、Rui Lu、Shuaiqi Duan、Shudan Zhang、Shulin Cao、Shuxun Yang、Weng Lam Tam、Wenyi Zhao、Xiao Liu、Xiao Xia、Xiaohan Zhang、Xiaotao Gu、Xin Lv、Xinghan Liu、Xinyi Liu、Xinyue Yang、Xixuan Song、Xunkai Zhang、Yifan An、Yifan Xu、Yilin Niu、Yuantao Yang、Yueyan Li、Yushi Bai、Yuxiao Dong、Zehan Qi、Zhaoyu Wang、Zhen Yang、Zhengxiao Du、Zhenyu Hou 和 Zihan Wang。ChatGLM：从 GLM-130B 到 GLM-4 All Tools 的大语言模型系列，2024。[12] Chaoqun He、Renjie Luo、Yuzhuo Bai、Shengding Hu、Zhen Leng Thai、Junhao Shen、Jinyi Hu、Xu Han、Yujie Huang、Yuxiang Zhang、Jie Liu、Lei Qi、Zhiyuan Liu 和 Maosong Sun。OlympiadBench：用于以奥林匹克级双语多模态科学问题推动通用人工智能发展的高难度基准，2024。URL https: //arxiv.org/abs/2402.14008。[13] Dan Hendrycks、Steven Basart、Saurav Kadavath、Mantas Mazeika、Akul Arora、Ethan Guo、Collin Burns、Samir Puranik、Horace He、Dawn Song 和 Jacob Steinhardt。使用 APPS 衡量编程挑战能力。NeurIPS，2021。[14] hiyouga。Math12k 数据集，2025。[15] Hanxu Hu、Xingxing Zhang、Jannis Vamvas、Rico Sennrich 和 Furu Wei。Quest：激励大语言模型生成难题，2025。URL https://arxiv.org/abs/2510.17715。[16] Naman Jain、King Han、Alex Gu、Wen-Ding Li、Fanjia Yan、Tianjun Zhang、Sida Wang、Armando Solar-Lezama、Koushik Sen 和 Ion Stoica。LiveCodeBench：用于代码的大语言模型全面且无污染评估。arXiv 预印本 arXiv:2403.07974，2024。[17] Lewis Tunstall Jia LI、Edward Beeching 等。NuminaMath TIR，2024。[18] Yuhang Lai、Chengxi Li、Yiming Wang、Tianyi Zhang、Ruiqi Zhong、Luke Zettlemoyer、Scott Wen tau Yih、Daniel Fried、Sida Wang 和 Tao Yu。DS-1000：用于数据科学代码生成的自然且可靠的基准。ArXiv，abs/2211.11501，2022。

[19] Yujia Li、David Choi、Junyoung Chung、Nate Kushman、Julian Schrittwieser、Rémi Leblond、Tom Eccles、James Keeling、Felix Gimeno、Agustin Dal Lago等。《使用AlphaCode进行竞赛级代码生成》。《科学》，378（6624）：1092–1097，2022年。[20] Haoxiong Liu、Yifan Zhang、Yifan Luo和Andrew Chi-Chih Yao。《通过迭代式问题编写增强数学文字题》，2024年。网址：https://arxiv.org/abs/2401.09003。[21] Thang Luong、Dawsen Hwang、Hoang H. Nguyen、Golnaz Ghiasi、Yuri Chervonyi、Insuk Seo、Junsu Kim、Garrett Bingham、Jonathan Lee、Swaroop Mishra、Alex Zhai、Clara Huiyi Hu、Henryk Michalewski、Jimin Kim、Jeonghyun Ahn、Junhwi Bae、Xingyou Song、Trieu H. Trinh、Quoc V. Le和Junehyuk Jung。《迈向稳健的数学推理》，2025。网址：https://arxiv.org/abs/2511.01846。[22] Chaitanya Manem、Pratik Prabhanjan Brahma、Prakamya Mishra、Zicheng Liu和Emad Barsoum。《Sand-math：利用大语言模型生成新颖、困难且实用的数学问题与答案》。arXiv预印本，arXiv:2507.20527，2025。[23] Qizhi Pei、Zhuoshi Pan、Honglin Lin、Xin Gao、Yu Li、Zinan Tang、Conghui He、Rui Yan和Lijun Wu。《ScaleDiff：扩展难题以推进高级数学推理》。arXiv预印本，arXiv:2509.21070，2025。[24] Qizhi Pei、Zhuoshi Pan、Honglin Lin、Xin Gao、Yu Li、Zinan Tang、Conghui He、Rui Yan和Lijun Wu。《ScaleDiff：扩展难题以推进高级数学推理》，2025年。网址：https://arxiv.org/abs/2509.21070。[25] 字节跳动 Seed。《Seed1.8 模型卡：迈向通用的现实世界自主能力》，2025年。[26] Vedant Shah、Dingli Yu、Kaifeng Lyu、Simon Park、Jiatong Yu、Yinghui He、Nan Rosemary Ke、Michael Mozer、Yoshua Bengio、Sanjeev Arora等。《人工智能辅助生成困难数学问题》。arXiv预印本，arXiv:2407.21009，2024。[27] Zhihong Shao、Peiyi Wang、Qihao Zhu、Runxin Xu、Junxiao Song、Xiao Bi、Haowei Zhang、Mingchuan Zhang、YK Li、Yang Wu等。《DeepSeekMath：推动开放式语言模型数学推理能力的极限》。arXiv预印本，arXiv:2402.03300，2024。[28] Guangming Sheng、Chi Zhang、Zilingfeng Ye、Xibin Wu、Wang Zhang、Ru Zhang、Yanghua Peng、Haibin Lin和Chuan Wu。《HybridFlow：一种灵活高效的基于人类反馈的强化学习框架》。载于《第二十届欧洲计算机系统会议论文集》，第1279–1297页，2025年。[29] Haoxiang Sun、Yingqian Min、Zhipeng Chen、Wayne Xin Zhao、Zheng Liu、Zhongyuan Wang、Lei Fang和Ji-Rong Wen。《挑战推理边界：面向大语言模型的奥赛级数学基准》，2025年。网址：https://arxiv.org/abs/2503.21380。[30]Hemish Veeraboina。《AIME 1983–2024年题集》，2023年。网址：https://www.kaggle.com/datasets/hemishveeraboina/aime-problem-set-1983-2024。[31] Shengbo Wang、Mingwei Liu、Zike Li、Anji Li、Yanlin Wang、Xin Peng和Zibin Zheng。《EvolMathEval：通过进化测试构建可演化的数学推理基准》，2025年。网址：https://arxiv.org/abs/ 2508.13003。[32] Yunhui Xia、Wei Shen、Yan Wang、Jason Klein Liu、Huifeng Sun、Siyue Wu、Jian Hu和Xiaolong Xu。《LeetCodeDataset：用于稳健评估和高效训练代码大语言模型的时间数据集》。arXiv预印本，arXiv:2504.14655，2025年。[33] Roy Xie、Chengxuan Huang、Junlin Wang和Bhuwan Dhingra。《对抗性数学文字题生成》。arXiv预印本，arXiv:2402.17916，2024年。[34] Roy Xie、Chengxuan Huang、Junlin Wang和Bhuwan Dhingra。《对抗性数学文字题生成》，2024年。网址：https://arxiv.org/abs/2402.17916。[35] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv等。《通义千问3技术报告》。arXiv预印本，arXiv:2505.09388，2025年。[36] yimingzhang。《ASDiv数据集》，2025年。[37] Charlie Zhang、Graham Neubig和Xiang Yue。《预训练、中期训练与强化学习对推理语言模型的影响》，arXiv预印本，arXiv:2512.07783，2025年。[38] Xueliang Zhao、Wei Wu、Jian Guan、Zhuocheng Gong和Lingpeng Kong。《PromptCoT 2.0：扩展提示词合成以增强大语言模型推理能力》。arXiv预印本，arXiv:2509.19894，2025年。

[39] Xinyue Zheng、Haowei Lin、Shaofei Cai、Zilong Zheng和Yitao Liang。《Unicode：一种生成高质量竞赛编程题目的框架》，2025年。网址 https://arxiv.org/abs/2510.17868。[40] Shang Zhou、Zihan Zheng、Kaiyuan Liu、Zeyu Shen、Zerui Cheng、Zexing Chen、Hansen He、Jianzhu Yao、Huanzhi Mao、Qiuyang Mang、Tianfu Fu、Beichen Li、Dongruixuan Li、Wenhao Chai、Zhuang Liu、Aleksandra Korolova、Peter Henderson、Natasha Jaques、Pramod Viswanath、Saining Xie和Jingbo Shang。《AutoCode：将大语言模型用作竞赛编程出题者》，2025年。网址 https://arxiv.org/abs/2510.12803。[41] Yubo Zhu、Dongrui Liu、Zecheng Lin、Wei Tong、Sheng Zhong和Jing Shao。《大语言模型早已知晓：通过隐藏表示估计大语言模型感知的题目难度》。载于《2025年自然语言处理经验方法会议论文集》，第1160–1176页，2025年。

附录

CoDiQ 流水线：案例研究

我们通过一个完整的工作流程展示 CoDiQ 流水线：从一个初始的简单问题开始，逐步提升难度，呈现成功升级的案例和失效模式。每个生成的问题都会经过可解性验证（附录 J）和难度评估（附录 I）。

A.1 初始问题

问题陈述： 统计数组 nums 中和为奇数的子序列数量，并返回对$10^9 + 7$进行模运算后的结果。

示例： 对于 nums = `[1,1,1]`，答案为 4（位置为：$\{0\}$、$\{1\}$、$\{2\}$、$\{0, 1, 2\}$ 的子序列，其和均为奇数）。

解法： 在 $O(n)$ 时间内使用简单动态规划跟踪和的奇偶性。

A.2 第 1 轮：受控升级

A.2.1 升级后的问题

统计同时满足以下三个条件的非空子序列数量：

1. 和为 **奇数**2. 长
度为 **偶数**3. 和
mod$3 = 1$

难度提升： 此次升级引入了多维状态跟踪，将动态规划状态空间从 2 个状态（和的奇偶性）扩展到 $2 \times 2 \times 3 = 12$ 个状态（和的奇偶性、长度奇偶性、和 mod 3 ）。

A.2.2 验证

- **可解性评分：** 0.90
- **时间复杂度：** $O(12n) \approx 1.2 \times 10^6$ 次运算，适用于 $n = 10^5$ （可行）
- **解密度：** $\sim 8\%$ 的子序列满足所有条件（非平凡）
- **可解性：** 通过
- **难度：** 提高

A.3 第2轮：进一步升级

A.3.1 升级后的问题

统计满足五个条件的子序列数量：

1. 和为**奇数**2. 长度为
偶数3. 和 mod$3 = 1$
4. 和 mod$5 = 25$. 长
度 mod$4 = 2$

数学简化： 根据中国剩余定理（中国剩余定理），条件 1、3 和 4 可以统一表示为：

$$\text{和} \equiv 1 \text{ (模 2)}, \quad \text{和} \equiv 1 \text{ (模 3)}, \quad \text{和} \equiv 2 \text{ (模 5)} \quad \Rightarrow \quad \text{和} \equiv 7 \text{ (模 30)}$$

有效状态空间变为 $30 \times 4 = 120$ 个状态。

A.3.2 验证

- **可解性评分：** 0.85
- **时间复杂度：** $O(120n) \approx 1.2 \times 10^7$ 次运算，$n = 10^5$（可接受）
- **解密度：** $\sim 0.83\%$（仍非平凡）
- **可解性：** 通过
- **难度：** 不变

A.4 第 3 轮：过度升级失败

A.4.1 升级后的问题

统计满足以下六个条件的子序列数量：

1. 和为**奇数**2. 和模$3 = 1$3. 和模$5 = 2$4. 和模$7 = 4$
5. 和模$11 = 6$6. 长度模$8 = 2$（确保长度为偶数）

根据中国剩余定理，条件 1–5 可合并为和 $\equiv c$（模 2310），其中某个常数为 c，由此得到包含$2310 \times 8 = 18,480$ 个状态的状态空间。

A.4.2 验证

- **可解性评分：** 0.65
- **可解性：** 失败
- **难度：** 提高

失败分析：**1. 计算不可行性**

- 时间复杂度：$O(18,480n) \approx 1.8 \times 10^9$ 次运算，用于 $n = 10^5$
- 超出竞赛编程的实际限制（通常要求在时间限制内完成 $\sim 10^8$–10^9 次运算）

2. 解空间坍缩（关键问题）

- 虽然这些约束通过中国剩余定理在数学上相容，但它们形成的解空间极其稀疏
- 随机子序列满足所有条件的概率：$\approx \frac{1}{2310} \times \frac{1}{8} = \frac{1}{18,480}$
- 有效子序列的期望数量：$18,480^{2^n}$
- 对于 $n \leq 14$：$\frac{2^{14}}{18,480} \approx 0.89 < 1$

- **实际影响：**对于 n较小至中等的典型输入，答案几乎总是0，从而使该问题在空洞意义上变得平凡

流水线终止：流水线在第 3 轮正确终止，丢弃 Q_3 并输出 $\{Q_0, Q_1, Q_2\}$。尽管理论难度有所提高，该问题仍因计算不可行性和解空间坍缩而无法求解，这证明了可解性验证在防止质量下降方面的有效性。

B CoDiQ 流水线：失败类型分析

为系统理解 CoDiQ 流水线的失效模式，我们对收集到的失败理由进行了全面的聚类分析。我们的分析采用三阶段层次方法：初始 K-means 聚类、关键词提取，以及结合人工精炼的层次合并。

B.1 聚类方法

阶段 1：K-means 预聚类。我们首先对失败理由描述应用 K-means 聚类，以获得数据的初始划分。这一预聚类步骤降低了计算复杂度，并对相似的失败模式提供了粗粒度分组。

阶段 2：关键词提取。对于从 K-means 得到的每个簇，我们使用 TF-IDF 加权提取代表性关键词。这些关键词作为语义特征，刻画每个簇内的主导失败模式，从而便于解释并支持后续的层次分析。

阶段 3：层次聚类与人工精炼。随后，我们利用提取出的关键词计算语义相似度，并对聚类质心进行层次聚类。最后，我们人工合并相关簇，并将较小的簇（样本数少于预定义阈值）与语义上最接近的簇合并。这种混合方法兼顾了计算效率与语义连贯性。

B.2 失效类别分布

表 6 对识别出的失效模式进行了分类。分析揭示了两种主要且截然不同的失效动态：**有效性缺陷**（无法求解）和**复杂度退化**（难度降低）。

无法求解的情形。 流水线失效大多源于问题表述存在根本性缺陷。具体而言，定义与信息缺失 和约束与逻辑冲突 共同构成了无法求解案例中的绝大多数。这表明，主要挑战不在于解析或格式处理（这类问题所占比例微不足道），而在于模型在生成过程中维持语义一致性和逻辑完备性的能力。

难度保持。 一个值得关注的发现是难度降低 类别占比很高（N =12,916）。在这些案例中，生成的问题仍然可解，但未能达到预期的认知要求。约束简化 和数值范围缩减 频繁出现，说明模型倾向于选择“稳妥”或简化的生成路径，无意中缩小了解空间，或移除了高质量数学推理所需的关键逻辑难点。

C 价值网络训练详情

C.1 数据集划分

我们通过从标准基准中选取样本 [8, 32] 作为简单 类，并从竞赛级数据集中选取样本 [13, 17] 作为困难 类，构建了一个标注数据集。我们将简单与困难比例设为 2:3，以优先识别具有挑战性的样本。我们将构建的数据集按 85:15 划分为训练集与测试集，以确保评估稳健。

表 6 CoDiQ 流水线中的失败案例分布

失败类型	失败子类型	数量
无法求解	定义与信息缺失	8,630
	约束与逻辑冲突	8,142
	计算上不可行	1,948
	实现细节缺失	1,926
	没有合适的算法/结构	1,285
	过于复杂	1,104
	需要特定能力	726
	解析与规则歧义	611
	数论约束	566
	其他	1,184
难度降低	约束简化	3,245
	数值范围缩减	2,890
	关键条件移除	2,654
	解空间收窄	1,987
	结构简化	1,456
	其他	684

C.2 训练数据

C.2.1 输入特征

对于训练数据，我们使用通义千问3-8B（非思考模式）来捕捉生成动态。我们将采样窗口定义为从问题的最后一个词元开始，延伸至 min(4096，L_r) 个生成词元。

在此窗口内，我们应用二次采样策略 来选择位置处的 K 隐藏状态（$K = 10$ 适用于窗口> 1024，否则为 $K = 8$）：

$$p_i = \left\lfloor |W| \cdot \left(\frac{i}{K-1} \right)^2 \right\rfloor, \quad i = 0, 1, \ldots, K-1. \tag{4}$$

C.2.2 输出标签

该策略会在生成起始阶段分配更高的采样密度，从而捕获建立推理路径所需的关键信息。为减小随机性，分数取 5 个独立采样轮次的平均值。

对于每个问题，我们使用 Qwen3-8B 生成回答，并根据最终答案的正确性分配一个二值标签 $y \in \{0, 1\}$ 。输入特征 x 通过应用于前 4096 个令牌的二次采样策略（公式 4）提取。

C.3 网络架构

价值网络采用轻量级多层感知机（MLP）实现，用于将高维隐藏状态（$d_{in} = 4096$）投影为标量正确性分数。该架构由初始投影层、层归一化、GELU 激活函数和最终回归头组成。

这种设置使网络能够通过加权二元交叉熵目标函数，最小化与正确性标签 y 之间的差异，从而仅根据早期阶段捕获的推理动态，有效估计成功生成的可能性。

表 7 价值网络的配置与性能评估。

表 8 超参数设置。

超参数	值
输入维度（d_{in}）	4096
隐藏维度	512
批量大小	512
学习率	1×10^{-4}
权重衰减	1×10^{-2}
随机失活	0.3
优化器	AdamW
调度器	StepLR（$\gamma = 0.8$）
最大轮次	30
数据划分	85% / 15%

表 9 留出测试集上的性能。

指标	值
准确性	72.52%
精确率	54.21%
召回率	95.62%
F1 分数	69.20%
ROC-AUC	**84.84%**
PR-AUC	65.77%

C.4 训练配置

模型使用 AdamW 优化器和阶梯学习率调度器进行训练。为应对类别不平衡，我们在损失函数中使用正类权重，并将其动态计算为负样本与正样本的比值。完整的超参数设置列于表8。

C.5 性能评估

我们在留出测试集（15% 数据划分）上评估训练好的价值网络。如表9所示，该模型的**ROC-AUC 达到 84.84%**，尽管数据集颇具挑战性，仍能稳健地将正确推理路径与错误推理路径区分开来，展现出强大的判别能力。

值得注意的是，我们的训练策略优先识别所有可能的正确答案。这体现在**95.62% 的高召回率**上，确保价值网络能够成功保留有效推理路径。尽管这种侧重覆盖率的做法由于类别加权训练固有的权衡而导致精确率适中（54.21%），但较高的 ROC-AUC 表明，预测分数能够有效地将正确生成结果排在更高位置，使模型适用于难度估计和筛选。

D CoDiQ-Bench 筛选标准

为确保基准的质量和可靠性，我们制定了三项主要的数据筛选标准：

可解性： 我们会验证每个问题是否定义明确且至少存在一个有效解，从而确保基准的有效性和公平性。

难度级别： 我们会评估难度级别是否适合预期的评测目的，并确保不同复杂度级别的分布均衡。

质量评估： 我们会进行严格的质量检查，确保所有入选问题在清晰度、正确性和相关性方面均达到可接受的标准。

E CoDiQ-Corpus 统计数据

我们采用 CoDiQ-Gen-8B，遵循 CoDiQ 流水线（第3.3节），将八个多样化的数学与编程数据集转化为更具挑战性的 CoDiQ-Corpus，其中包含约 44,453 条难度从简单到困难逐步递进的问题序列。详细分布见表10。

表 **10** CoDiQ-Corpus 数据集统计。

数据集	问题词元长度			平均轮次	类别	序列
	最小值	最大值	平均值			
Math12K [14]	38	7,829	995.4	4.7	Math	11,764
GSM8K [8]	52	6,896	1,093.7	4.5	Math	8,685
SVAMP [6]	172	3,992	971.3	3.3	Math	804
ASDiv [36]	55	4,703	1,013.1	4.7	Math	1,480
CodeAlpaca20K [4]	70	7,174	1,106.1	3.8	Code	17,845
LeetCodeDataset [32]	254	4,365	1,281.0	3.8	Code	2,027
MBPP [2]	52	3,440	1,000.4	3.4	Code	876
DS-1000 [18]	192	4,138	1,240.7	3.2	Code	972
总计	38	7,829	1,073.0	4.2	-	44,453

F 扩展趋势分析详情

本节呈现涵盖所有已评估模型的完整扩展趋势分析。图 6 展示了所有长链式思维模型在直接提示与 CoDiQ 提示两种设置下、历经 8 个生成轮次的难度与可解性扩展的完整结果。

完整结果显示，所有模型都呈现出一致的扩展模式：（1）推理计算增加与问题难度提高相关；（2）难度与可解性之间存在权衡，较大的模型能够在这两个指标之间保持更好的平衡。

G CoDiQ-Corpus 质量标准

我们制定了严格的标准，用于评估 CoDiQ-Corpus 中问题的质量和可解性。三位博士级领域专家按照这些标准化指南，独立评估了随机抽取的 300 道问题：

G.1 信息完整性

- **充足参数**：所有必要的数值、变量和约束均已明确提供。
- **明确目标**：问题目标清晰明确，定义清楚。
- **完整上下文**：不存在截断或缺失的问题陈述。

G.2 逻辑一致性

- **不矛盾条件**：所有给定约束彼此一致。
- **有效前提**：对于逻辑问题，前提足以支持结论。
- **可行解**：该问题至少存在一条有效解题路径。

G.3 问题定义明确性

- **可确定答案**：答案可以被唯一确定，或限定在合理范围内。
- **适当范围**：问题复杂度与其所述领域和难度级别相匹配。
- **标准表述**：遵循常规的数学或逻辑记法。

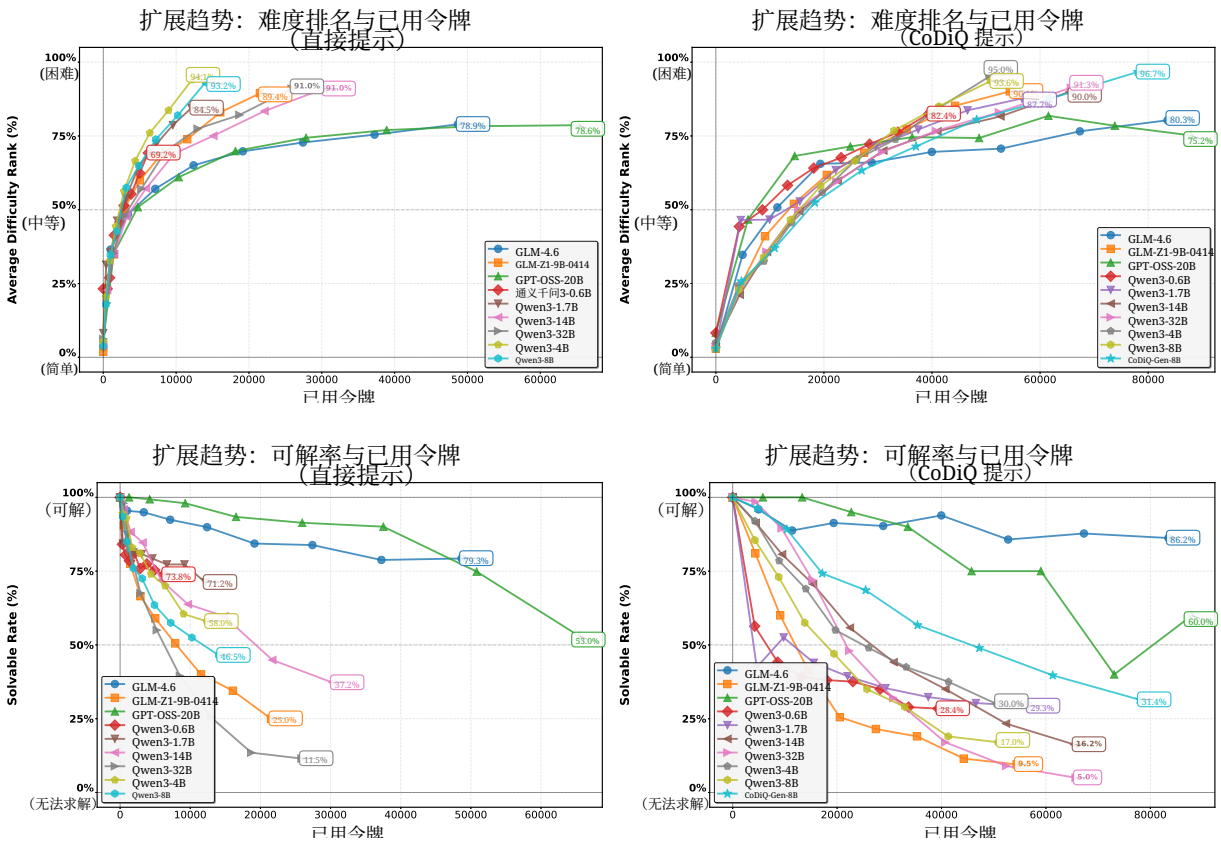

图 6 CoDiQ-Bench 上的完整缩放分析。 所有受评估长链式思维模型在 8 轮中生成问题的归一化平均难度排名（上排）和可解率（下排），左侧使用直接提示，右侧使用 CoDiQ 提示。排名越高表示问题难度越高；可解率越高表示问题质量越好。

G.4 评估协议

每位专家给出二元可解性标签（可解/无法求解）及置信度分数。仅当至少两名专家达成一致时，问题才会被标记为**可解**。分歧通过讨论解决。标注者间一致性（Fleiss' κ）达到 0.78，表明存在实质性共识。

H 课程学习详情

H.1 课程学习的训练数据选择

为验证 CoDiQ-Corpus 对课程学习的有效性，我们仔细筛选出具有渐进难度结构的问题序列。具体而言，我们从 CoDiQ-Corpus 中抽取了 480 个问题序列，每个序列的长度为 $|S| \geq 3$，构成课程学习数据集：

$$\mathcal{D}_{\text{curriculum}} = \{S_n\}_{n=1}^{480}, \quad |S_n| \geq 3 \tag{5}$$

对于每个具有渐进难度的序列 $S_n = \{q_0^n, q_1^n, q_2^n, \ldots, q_{|S_n|-1}^n\}$，我们构建了三个复杂度递增的训练阶段：

- **第1级（L1）：** 包含每个序列中的所有初始问题 q_0^n，代表各难度递进的起点。

- **第2级（L2）：** 随机抽取每个序列中间位置的一个问题 $\{q_1^n, q_2^n, \ldots, q_{|S_n|-2}^n\}$，以呈现中间阶段的复杂度。

- **第3级（L3）：** 包含每个序列中的所有最终问题，代表各难度递进序列中的最高难度级别。 $q_{|S_n|-1}^n$

形式化地说，数据选择策略定义如下：

$$\mathcal{L}_1 = \{q_0^n \mid S_n \in \mathcal{D}_{\text{curriculum}}\} \tag{6}$$
$$\mathcal{L}_2 = \{\text{random}(\{q_i^n\}_{i=1}^{|S_n|-2}) \mid S_n \in \mathcal{D}_{\text{curriculum}}\} \tag{7}$$
$$\mathcal{L}_3 = \{q_{|S_n|-1}^n \mid S_n \in \mathcal{D}_{\text{curriculum}}\} \tag{8}$$

这种设计确保了清晰的难度递进：难度($\mathcal{L}_1$) $<$ 难度($\mathcal{L}_2$) $<$ 难度($\mathcal{L}_3$)。各级别的样本分布遵循以下比例 $|\mathcal{L}_1|:|\mathcal{L}_2|:|\mathcal{L}_3| = 2:2:1$，通过在训练期间重复使用 $\mathcal{L}_1$和 $\mathcal{L}_2$ 来实现，从而平衡模型对不同难度级别的接触。该比例旨在遵循课程学习原则，在进阶到更具挑战性的问题之前，确保模型获得充分的基础学习 [3]。

对于基线模型 Qwen3-RL-4B，我们使用原始的、未经转换的数据集（应用 CoDiQ 流水线之前）作为训练数据，同时保持训练样本总数不变，以确保比较公平。这样可以隔离渐进式难度转换对模型性能的影响。

训练计划：模型分三个阶段依次进行训练：

1. CoDiQ-L_1-4B：在 $\mathcal{L}_1$ （起始级别）上训练 2. CoDiQ-L_2-4B：从 CoDiQ-L_1-4B 初始化，并在 $\mathcal{L}_2$ （中级）上进一步训练 3. CoDiQ-L_3-4B：从 CoDiQ-L_2-4B 初始化，并在 $\mathcal{L}_3$ （高级）上进一步训练

H.2 奖励信号设计

我们通过提示通义千问3-32B 作为专家评估器，设计了一个多维奖励函数来评估答案质量。奖励信号 $r \in [0, 1]$ 基于四个关键维度计算：

评估维度：

- **问题解决度 (s_{pr})**：衡量答案对问题各个方面的回应是否完整（0.0-1.0）。

- **推理正确性 (s_{rc})**：评估推理过程的正确性与连贯性（0.0-1.0）。
- **信息完整性 (s_{ic})**：评估是否包含所有必要的信息、步骤和解释（0.0-1.0）。

- **准确性 (s_{acc})**：衡量事实正确性、计算准确性和概念清晰度（0.0-1.0）。

奖励函数将这些维度与经过仔细调校、针对高难度数学推理任务进行优化的权重相结合：

$$r = w_{\text{pr}} \cdot s_{\text{pr}} + w_{\text{rc}} \cdot s_{\text{rc}} + w_{\text{ic}} \cdot s_{\text{ic}} + w_{\text{acc}} \cdot s_{\text{acc}} \tag{9}$$

其中，默认权重设置如下：

$$w_{\text{pr}} = 0.20, \quad w_{\text{rc}} = 0.35, \quad w_{\text{ic}} = 0.25, \quad w_{\text{acc}} = 0.20 \tag{10}$$

此配置侧重于推理质量（35%）和信息完整性（25%），这两者对于复杂问题求解至关重要。评估提示词要求 Qwen3-32B 使用连续分数独立评估各个维度，并以 JSON格式返回结果。对于以下情况，会进行特殊处理：

边界情况，例如正确识别不可解问题；即使没有给出数值解，也会获得较高的问题解决评分（0.8-1.0）。

为确保评估质量，我们会对返回的分数进行自动验证，设置重试机制（最多尝试 3 次），并通过文本截断处理较长的输入（问题最多 4096 个令牌，答案最多 16384 个令牌）。评估器返回的置信度分数有助于识别不确定的评估结果，以便进行人工审核。

I 大语言模型排名指令

大语言模型排名指令

你是一位评估问题难度的专家。请根据以下方面评估问题：
1. 知识复杂度：所需概念的数量和深度 2. 认知负荷：所需的推理层次和抽象思维能力 3. 计算复杂度：涉及的步骤和计算 4. 陷阱与常见错误：问题中的隐蔽陷阱 5. 综合应用能力：跨领域知识应用的要求

你的任务是按难度级别对问题分组，并按从最简单到最困难的顺序排列各组。

重要提示： 难度级别相同的问题应归为一组。

请仔细分析每个问题，并按难度级别分组返回。

输出格式要求：

- 仅返回包含以下两个字段的有效 JSON 对象：
- **结果**：一个列表的列表（组），每个列表包含难度级别相同的问题索引
- **理由**：一个字符串列表，其中理由 [i]解释了为什么结果 [i] 中的问题具有相同的难度

- 两个数组中的分组都应按从最简单到最困难的顺序排列
- "结果"数组和"理由"数组的长度必须相同
- 使用从零开始的索引，并与输入顺序一致

示例输出格式：

```
{"结果": [[1, 3], [0], [2, 4]],"理由": ["两者都只需进行基本算术运算，并通过单步推理得出结果","运用中间概念进行多步代数运算","综合运用高级概念，并采用不明显的策略进行复杂整合"]}
```

这意味着：

- 问题 1 和 3 最简单（第 0 组）——它们都只涉及基本算术和单步推理

- 问题 0 的难度为中等（第1组）——它需要运用中间概念进行多步代数运算
- 第2组的问题2和问题4最困难——它们都要求复杂整合高级概念，并运用不明显的策略

重要：
- 每个推理字符串都应解释将相应组中所有问题联系起来的共同难度特征
- 确保 reason[i] 与所有组的 result[i] 相对应
  请按难度级别对以下问题进行分组，并将各组从最简单到最困难排序：{questions}

请以以下格式将结果返回为 JSON 对象：

{"结果": [[最简单组的索引], [下一组的索引], ...], "理由": ["第 0 组的推理", "第 1 组的推理", ...]}

J 可解性检查说明

可解性检查说明

你是一位擅长分析数学和逻辑问题的专家。你的任务是判断给定的问题是否可解。

问题被视为**可解**，如果：
1. 提供所有必要的信息和条件 2. 问题定义明确且目标清晰 3. 答案可确定（即使问题很复杂） 4. 约束条件一致（不存在矛盾）

问题被视为**无法求解**，如果：
1. 缺少关键信息或参数 2. 包含矛盾条件 3. 问题陈述含糊不清 4. 要求提供无法根据给定数据确定的信息 5. 问题不完整或被截断

重要指南：
- 在判断时要严格但合理
- 考虑一个理性的人是否能够利用给定信息解决该问题
- 对于数学问题，检查是否提供了所有必要的数值
- 对于逻辑问题，请验证前提是否足以得出结论

输出格式要求：
- 仅返回有效的 JSON 对象

- 必须恰好包含以下字段:
  – "solvable": 布尔值（true/false）– "confidence": 数字（0.0-1.0，表示你对判断的置信度）– "reason": 字符串（用英文简要说明，最多 200 个字符）– "missing_info": 字符串列表（缺少哪些信息；如果可解，则为空列表）

输出示例:

{" solvable ": true , " confidence ": 0.95 , " reason ": "已提供所有必要参数，问题定义明确" , " missing_info ": [] } {" solvable ": false , " confidence ": 0.85 , " reason ": "缺少计算圆面积所需的半径值" , " missing_info ": [" radius "] }

K 直接提示说明

直接提示说明

 # 题目难度升级生成器
任务描述
你是一位竞赛编程题创作专家。你的任务是基于给定的问题，创作一个难度显著更高、达到竞赛级别的版本。

输入
原始问题:
{original_problem}

输出格式
仅返回升级后的新问题，不要添加其他内容。

[在此填写升级后的竞赛编程题]

L CoDiQ 提示的说明

CoDiQ 提示的说明

 # 题目难度升级生成器
任务描述
你是一位竞赛编程题专家。你的任务是对给定题目进行改编，通过有策略地加入难度要素，设计出难度显著更高的竞赛级版本，以考查更深入的理解和更复杂的推理能力。

设计标准（强制质量检查）
为确保 升级后的问题要达到竞赛水准，必须严格遵循以下原则:

1. **深度综合**：难度要素必须与原始逻辑自然交织。解题过程应像是在应对一个连贯完整的挑战，而非“拼凑”而成。 2. **多步推理**：解题至少需要 2 到 3 次非平凡的逻辑跳跃。解题者必须先推导出引理或中间状态，才能应用标准算法。 3. **禁止平凡升级**：避免仅将 N 增大到 10^5，而逻辑仍保持为 $O(N)$ 的情况。升级必须迫使复杂度类发生变化（例如，从贪心转向网络流，或从模拟转向矩阵快速幂）。 4. **伪装与抽象**：（如适用）通过独特的故事背景或抽象的数学设定隐藏核心定理或数据结构。切勿明确指出所需算法。**## 难度要素库（选择 1 至 2 个不同要素） ### 类别 A：维度与约束 适用于：**具有简单朴素解法的数组/序列/树问题（例如 $O(N)$、 $O(N^2)$ 或 $O(N^3)$）。**不适用情况：**原始问题已经要求对数或次线性复杂度。**说明：**扩大数据规模或维度，使简单模拟或暴力解法失效。**核心策略：** 1. **识别**朴素复杂度（例如 $O(N)$、 $O(N^2)$ 或 $O(N^3)$）。2. **施加**约束，迫使采用更优的复杂度类（例如 $O(\log N)$ 或 $O(N \log N)$）。3. **引入**动态更新、高维空间或多种查询类型，打破线性扫描。

示例：

[{ " original ": " 给定一个大小为 N（N≤1000）的数组，求区间 [L , R]内元素的和。" , " upgrade ": " 给定一个大小为 N（N≤10^5）的数组，处理 M（M≤10^5）次运算：1. 将区间 [L , R] 内的元素都加上 V。2. 查询区间 [L, R]内元素的和。（需要使用带懒标记的线段树）" }, { " original ": " 给定一个网格，求避开障碍物从 (0 ,0) 到 (R ,C) 的最短路径。" , " upgrade ": " 给定一个网格，其中的障碍物会在特定的时间区间内按模 K 周期出现和消失。求最短路径。（需要在状态空间 (x , y , time %K) 上进行广度优先搜索）" }, { " original ": " 求数组中的最大值。" , " upgrade ": " 给定一棵有 N 个节点（N≤10^5）的树，支持路径更新（将路径 u -v 上所有节点的值增加 V）和路径最大值查询。（需要使用重链剖分）" }, {

" original ": " 给定一组点，求距离最近的两个点。" , " upgrade ": " 给定三维空间中的一组点，求满足任意两点间的曼哈顿距离均为 >D 的最大子集的大小。（需要使用坐标变换 + 数据结构）" }, { " original ": " 检查字符串 S 是否包含模式 P。" , " upgrade ": " 给定文本 S 和 K 个模式。支持动态插入新模式，并查询是否有任一模式出现在 S 中。（需要使用 Aho - Corasick 自动机或后缀结构）" }]

类别 B：

数学抽象

最适合： 可以转化为数学结构的问题，例如具有明确规律的模拟问题或迭代问题。

避免用于： 问题已聚焦于高级专业定理或复杂数据结构。

描述： 将程序性或描述性问题转化为形式化模型，例如运用数论、组合数学或博弈论。

核心策略：

1. **增加**约束以突破计算能力的限制（例如 $N \geq 10^{18}$），使简单的迭代或模拟变得不可能。2. **迫使**人们发现底层结构，例如闭式公式、递推关系或不变量性质。3. **引入**形式化约束（例如模运算、坐标系），要求进行严谨的数学建模。

反面模式： 单纯增大 N，却不确保存在数学洞见，是不可取的。**示例：**

[{ " original ": " 模拟细菌每小时数量翻倍的过程。求第 N 小时的数量（N≤50）。", " upgrade ": " 细菌遵循复杂的生长规则 F(n) = a*F(n -1) + b*F (n -2)。求第 N 小时的数量（N≤10^18），并对 10^9+7取模。（需要矩阵快速幂）" }, { " original ": " 给定 N 个物品，有多少种选取 K 个物品的方法？ " , " upgrade ": " 给定 N 个物品，并附有特定的颜色约束，计算选取 K 个物品的方法数，并对 10^9+7 取模，其中 N 最大可达10^9。（需要卢卡斯定理或生成函数）" },

{ " original ": " 两名玩家轮流移除 1 -3 颗石子。谁会获胜？ " , " upgrade ": " 在一张有 N≤10^5 个节点的图上进行游戏。令牌沿边移动。无法移动的玩家输。图中存在环。（需要图上的博弈论／Sprague-Grundy 理论及环处理）" }, { " original ": " 计算两个数的最大公约数（GCD）。" , " upgrade ": " 计算所有 1 ≤i、j ≤ N 的最大公约数之和，其中 N≤10^7。（需要欧拉函数／莫比乌斯反演）" }, { " original ": " 给定多边形的整数坐标，求其面积。" , " upgrade ": " 给定平面上的 N 条直线，精确求出它们并集区域的面积。处理平行直线和共点直线。（需要积分逻辑或应用格林定理）" }]

类别 C:
逆向与构造

最适用于： 具有明确定义的算法、且“输入 → 算法 → 输出”流程清晰的问题。**避免时：** 原始问题的核心挑战已处于“设计/构造”阶段，而非“计算/处理”阶段（即缺乏可供逆向的标准算法的问题）。

描述： 不是要求某个过程的结果，而是要求能够产生特定结果的输入。

核心策略：

1. **反转**问题方向：从“给定 X，求 Y”变为“构造 X 使得 Y 成立”。2. **要求**理解结构性质（例如，是什么使一个图具有特定的网络流？）。3. **添加**多重约束，使构造并非易事。

示例：

[{ " original ": " 给定一个图，找出从 A 到 B 的最短路径。" , " upgrade ": " 构造一个具有 N 个顶点和 M 条边的图，使得从 1 到 N 的最短路径长度恰好为 L，且最小生成树的权重恰好为 W。" }, { " original ": " 使用快速排序对一个数组进行排序。" , " upgrade ": " 构造一个大小为 N 的排列，使标准快速排序实现（以第一个元素作为枢轴）达到其

最坏情况下的 O(N^2) 时间复杂度。" }, { " original ": " 给定一棵二叉树，输出其先序遍历。" , " upgrade ": " 给定先序遍历和后序遍历，重建所有可能的二叉树。确定解是否唯一，或统计满足条件的二叉树数量。" }, { " original ": " 检查一个字符串是否为回文。" , " upgrade ": " 构造一个长度为 N 的字符串，其中恰好包含 K 个不同的回文子串。证明当 K 超过某个界限时，不存在这样的字符串。" }, { " original ": " 求网络中的最大流。" , " upgrade ": " 给定所需的最大流值 F，在每条边均受容量约束的条件下，构造一个以最少边数实现该流值的网络。" }]

D类：
状态爆炸
适用于： 具有简单、多项式动态规划状态且可加以扩展的问题。
避免使用场景： 原始状态空间已经呈指数级（例如旅行商问题）；增加维度会导致计算上不可行。

描述： 通过扩展状态空间，加入复杂的依赖关系或历史记录要求，这需要采用高级动态规划或网络流。

核心策略：
1. **重新定义**状态：从简单状态（例如“dp[i]”）转向复合型、多维状态（例如，为集合添加指数级“掩码”，或为约束添加多项式“余数”）。2. **引入**依赖于过去选择或特定历史的约束（例如，“不能访问 k 步之前访问过的节点”，这需要使用滑动窗口或历史状态）。3. **添加**多个彼此正交、必须同时跟踪的限制（例如数量、奇偶性或模性质）。

反模式： 只是添加彼此不相互作用的变量。新增维度必须从根本上改变递推逻辑或转移依赖。

示例：

[{ " original ": " 爬楼梯，每次走 1 或 2 步。有多少种方法？ ", " upgrade ": " 用 1×2 多米诺骨牌覆盖一个 3×N 网格。有多少种方法模运算10^9+7？（需要轮廓线动态规划 / 状态压缩动态规划来跟踪跨 -

截面状态)" }, { " original ": " 背包问题：在重量限制 W 下的最大价值。" , " upgrade ": " 树上背包：每个节点都有价值 / 重量。选择节点使价值最大，且任意两个被选节点不相邻，总重量 ≤W。（需要树形动态规划 + 背包维度）" }, { " original ": " 数组中的最长递增子序列。" , " upgrade ": " 统计长度为 N、且最长递增子序列长度恰好为 K 的排列的数量。（需要结合杨表或 RSK对应的动态规划）" }, { " original ": " 求遍历网格的最小代价。" , " upgrade ": " 求借助 K 个“电池”越过障碍物来遍历网格的最小代价；充能取决于网格值模运算 M，且单元格必须按顺序解锁。（状态：位置 + 电池_数量 +模运算_状态 + 锁_掩码）" }, { " original ": " 两个字符串之间的编辑距离。" , " upgrade ": " 给定字符串 A 和 B，求长度为 L 的字符串 S 的数量，使得编辑距离 (A , S) ≤K 且编辑距离 (B , S) ≤K。（需要动态规划套动态规划 / 自动机动态规划）" }]

E类：
定理伪装
最适合： 能够映射到经典高级算法、但出现在无关或抽象领域中的问题。

应避免： 原始问题明确提及该算法或数据结构。
描述： 将精妙的算法核心隐藏在叙事或替代性数学结构之后，从而误导直觉。

核心策略：

1. **将问题映射到一种知名且非平凡的算法（例如网络流、线性基、生成函数或高级数据结构）**。 2. **删除**所有暗示解法的专业术语和明确约束。 3. **构造**一个“烟幕”式叙述，引导人们采用直观但次优的方法（例如贪心、简单动态规划或朴素模拟）。 4. **确保**表面问题与隐藏定理之间的联系需要深刻的结构性洞察。

示例：

[{ " original ": " 求一个人最多能参加多少场会议。" , " upgrade ": " 给定一组区间，用最少的颜色为它们着色，使得任何重叠的区间颜色都不同。有些区间是“贵宾”区间，必须使用指定的颜色。（伪装成贪心问题，实则是区间图着色 / 带约束的狄尔沃斯定理）" }, { " original ": " 能否将一个数组划分成两个元素和相等的子集？" , " upgrade ": " 给定一个图，判断它是否为二分图。如果不是，则删除最少的边，使其变为二分图，同时保持连通性。（伪装成图论问题，实则是奇圈横截集 / 最大割的变体）" }, { " original ": " 将任务分配给工人，使总耗时最短。" , " upgrade ": " 有 N 个工人和 M 项任务。每对（工人，任务）都有相应的成本。部分工人之间存在冲突，不能同时工作。求最小总成本。（伪装成带冲突处理的最小费用最大流）" }, { " original ": " 求最多能选出多少个彼此兼容的区间。" , " upgrade ": " 给定一个圆及圆上的 N 条弦，每条弦由其端点表示。求最多能选出多少条互不相交的弦。（伪装成圆图动态规划或与卡特兰数相关的结构）" }, { " original ": " 在数组中找出一个出现次数超过 N/2 的数。" , " upgrade ": " 给定一个包含 N 个元素的数据流，使用 O(K) 空间并仅遍历一遍，找出所有出现次数超过 N/K 的元素。处理 K 最大为`10^3`。（伪装成博耶-摩尔投票算法的推广 / 米苏拉-格里斯算法）" }]### F 类：边界情况与严谨性工程 最适合：逻辑简单但实现细节棘手的问题。**避免用于：**原始问题在技术性或实现复杂度方面已经很高的情况。**描述：**聚焦于逻辑陷阱、几何精度、棘手的边界情况，或需要通过形式化证明来确保正确性的问题。**核心策略：**

1. **针对** 标准数据类型的弱点（如浮点精度、整数溢出、空集）。2. **引入** 退化情形（如非连通图、共线点、零分母）。3. **要求** 严格的数学证明（如“交换论证”），以证明在直觉失效时采用某种策略的合理性。4

范围。**示例：** [{ " original ": " 给定 3 个点的坐标，计算三角形的面积。" , " upgrade ": " 给定平面上的 N 条直线（N≤`10^5`）。求它们相交形成的有限区域的面积。处理平行直线、无穷远直线和精度误差。（需要使用半平面相交算法，并进行严格的边界处理和误差比较）" }, { " original ": " 判断图中是否存在从 A 到 B 的路径。" , " upgrade ": " 判断 A 是否可达 B。图中可能包含自环、重边，且最多由 `10^5` 个不连通分量组成。回答 `10^5` 次在线查询。处理 A=B 的情况，以及 A 或 B 不存在的情况。（测试边界情况：连通性、带路径压缩的并查集、查询验证）" }, { " original ": " 将 A 除以 B。" , " upgrade ": " 将 A 除以 B，其中 A 和 B 是表示整数的字符串，长度最多为 `10^1000`。处理 B=0、负号、前导零，并输出商和余数。证明除法算法的正确性。（需要实现大整数，并进行 `+`全面的边界情况处理）" }, { " original ": " 贪心：总是选用可用的最大面额硬币来找零。" , " upgrade ": " 给定一个非标准硬币系统（例如 1、3、4），求贪心策略无法找到最少硬币数的最小值 V。证明对于所有小于 V 的数值，贪心策略都是最优的。（需要找出算法本身的边界情况 `+` ，并使用交换论证）" }, { " original ": " 按持续时间对任务数组排序。" , " upgrade ": " 给定 N 个任务及其处理时间和衰减率。每个任务在等待期间都会以不同速率损失价值。找出一种

. **Add** precision constraints（例如，回答准确度 `10` to `10`）or handle ext

处理顺序，使总衰减最小。使用交换论证证明你的排序策略。（需要通过形式化证明推导出正确的比较器；简单排序是错误的）" }]

构造协议（内部思考过程）

1. **分析原始问题**：识别朴素解法及其复杂度。2. **选择类别**：从上方库中选择 1-2 个最契合该问题潜力的类别。3. **应用核心策略**：使用所选类别中定义的“核心策略”，重新设计问题的约束与目标。4. **审查**：对照“设计标准”进行检查。它是否需要多步推理？定理是否经过伪装？5. **最终输出**：使用标准竞赛编程格式，清晰撰写问题陈述。

输入

原始问题：

{original_problem}

输出格式

只返回新的升级后的问题，不要返回其他任何内容。

[在此填写你的升级后的竞赛编程题]