Break-The-Chain：通过代码生成中的对抗性提示来分析大型语言模型中的推理失败

Jaechul Roh, Varun Gandhi, Shivani Anilkumar, Arin Garg
{ jroh, vgandhi, sanilkumar, aringarg } @umass.edu

Abstract

大型语言模型（LLMs）在需要复杂推理的任务中取得了显著的成功，例如代码生成、数学问题解决和算法合成——尤其是在推理标记和连锁思维提示的帮助下。然而，一个核心问题仍然存在：这些模型是否真正具备推理能力，还是仅仅利用了浅层统计模式？在本文中，我们通过引入一套语义忠实但具有对抗性结构的提示扰动，系统地研究了推理 LLM 的稳健性。我们的评估——涉及 700 个来自 LeetCode 风格问题的扰动代码生成——应用了叙事重构、无关约束注入、示例重排和数值扰动等转变。我们观察到，虽然某些修改严重降低了性能（准确率下降最多达-42.1 %），但另一些修改却出乎意料地提高了模型准确性，最高可达 35.3 %，表明模型不仅对语义敏感，而且对表面提示动态亦然。这些发现揭示了当前推理系统的脆弱性和不可预测性，强调了对推理校准和提示稳健性采取更具原则性的方法的必要性。我们发布了我们的扰动数据集和评估框架 [1]，以促进对可信且有韧性的 LLM 推理的进一步研究。

1 引言（问题陈述）

大型语言模型（LLM）近年来迅速发展，在多种任务中展示了令人印象深刻的能力——从自然语言理解和翻译到代码生成和复杂推理（Brown et al., 2020; Chowdhery et al., 2022; OpenAI, 2023; Bai et al., 2023; Chen et al., 2021）。通过架构上的改进和诸如指令微调、思维链（CoT）提示（Wei et al., 2022）以及来自人类反馈的强化学习等技术，现代 LLM 在曾被认为需要人类水平智能的基准测试上实现了最先进的性能。

然而，尽管大型语言模型在广度上表现卓越，它们仍然容易受到对抗性输入的攻击。措辞上的微小变化——通常在语义上无关紧要——可以显著改变模型的行为，尤其是在代码生成等敏感应用中。这种脆弱性不仅仅是表面现象：细微的提示扰动可能导致模型忽略逻辑、遗漏约束，甚至生成不安全的代码。正如对 DeceptPrompt (Wu et al., 2023) 和提示注入攻击的先前研究所展示的，这些模型通常依赖于提示中的表面模式，从而使它们在面对现实世界输入的变化时变得脆弱。

最近的努力强调了大型语言模型的新兴推理能力。通过像连续提示这样的技术，模型现在可以处理涉及多步骤计算、逻辑推断和算法合成的任务。这些发展激发了希望，即大型语言模型可能正从随机的模仿者演变为真正的推理者。然而，关键问题仍然是：大型语言模型是真正进行推理，还是仅在提示与熟悉的模板匹配时模拟推理？这种区别至关重要，尤其是在评估语言或上下文变化下的稳健性时。

为了严格研究这个问题，Mirzadeh 等人（2024）提出了 GSM-Symbolic（（Mirzadeh et al., 2024））, 这是一个基准，它通过符号和语义转换扰动数学词问题，同时保持其逻辑结构不变。其结果显示，在良性修改下准确性显著下降，这表明即使任务语义保持不变，推理性能对表面形式非常敏感。

在这项工作中，我们将这一研究方向扩展到代码生成领域。我们引入了 BREAK-THE-CHAIN，这是一个用于评估大型语言模型（LLMs）推理鲁棒性的系统框架，采用的是语义对齐但具有对抗性的提示扰动。我们的扰动涵盖了一组多样的变换，包括讲故事、游戏化、领域转换、干扰性约束和否定等，这些变换在叙述结构、词汇框架和语义漂移方面各不相同。至关重要的是，每种变换都保留了原始任务的核心逻辑，使我们能够隔离表面层次变化对推理准确性的影响。

如图 1 所示，我们的方法揭示了大型语言模型在这些受控扰动下行为的显著差异。一些模型在低保留改写中崩溃，而另一些在叙述支架下反而出现了悖论性改进。与之前专注于清晰准确性的基准不同，我们的框架探查了大型语言模型的认知稳定性——强调了需要进行考虑

[1] https://github.com/jrohsc/Break-the-Chain

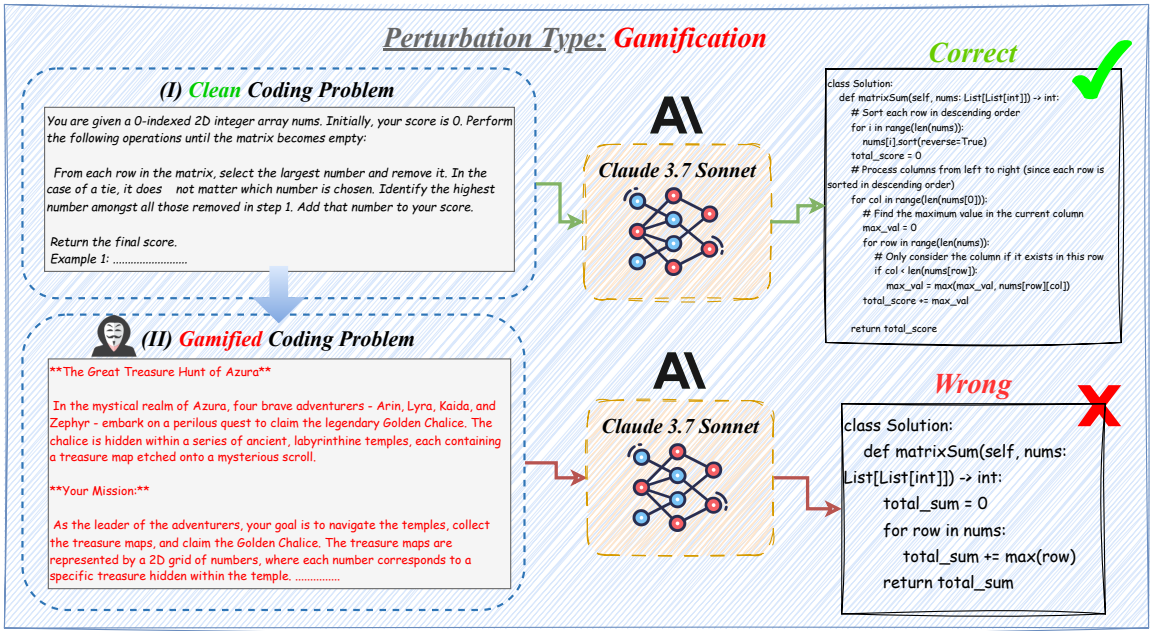

Figure 1: Break-the-Chain 概述

语言多样性、对抗意图以及与人类一致提示的稳健性评估。

在这篇论文中，我们成功实现了六种不同的扰动方法——数值扰动、语义子句注入、叙事法、游戏化、领域转换和否定。在现有的代码生成任务基准 (Chen et al., 2021; Austin et al., 2021) 中，我们将这些方法应用于从 LiveCodeBench 数据集中精心挑选的 100 个 LeetCode 风格的问题，(Jain et al., 2024)，生成了 700 个扰动实例。我们使用 Pass@1 (Jain et al., 2024) (改编自 LiveCodeBench) 评估了九个最先进的 LLM，通过基于编译的执行和人工检查来评估代码正确性。我们的结果显示，模型在干净提示上的表现范围从 95.0 % (Gemini-2.5-Flash) (DeepMind, 2024) 至 17.0 % (DeepSeek-Coder-1.3B) (Guo et al., 2025)，而一些扰动如叙事法则提高了特定模型的准确性 (例如，LLaMA-3.1-Instruct 增加了 23.5 % (Meta, 2024))。相反，诸如否定这样的低保留攻击在若干情况下使性能下降了超过 50 %。我们进行了一项详细的消融研究，并引入了语义扰动鲁棒性作为一种新的评估维度，以表征模型对良性语言变化的敏感性。

这项工作的相关性体现在两个方面。首先，随着大型语言模型 (LLMs) 逐渐整合到开发工作流程和编码助手中，确保其在各种不完美输入下的可靠性变得至关重要。其次，对于更广泛的自然语言处理 (NLP) 社区，我们的研究结果提供了一种新视角来评估模型的泛化能力——不仅在准确性方面，还在语义扰动下的行为一致性方面。我们相信这个框架可以作为未来关于可信和可解释的大型语言模型推理研究的基础。

2 相关工作

在评估和确保大型语言模型 (LLMs) 真正推理能力的挑战中，尤其是在代码生成等复杂任务上，已经从多个角度进行研究。尽管 LLMs 在多个自然语言处理任务上表现出色，但在确保其稳健性、可靠性以及实际推理质量的深度上仍存在显著挑战。我们的工作 BREAK-THE-CHAIN 基于并扩展了先前研究的三个关键领域，这些领域为我们研究对抗性提示扰动下的推理失败提供了信息：LLMs 中的推理性质，基于 LLM 的代码生成的具体进展和评估，以及针对 LLM 推理攻击的现有研究。

2.1 大语言模型推理

在大型语言模型 (LLM) 中，推理包括诸如数学问题求解、代码生成和逻辑推理等分析任务。虽然传统模型通常依赖于模式识别，但增强推理的模型越来越多地使用推理令牌和思维链 (Chain-of-Thought, CoT) 提示技术 (Wei et al., 2022)。推理令牌被设计为在训练过程中引导 LLM 进行更结构化推理的显式标记，这种方法在处理复杂推理挑战方面 (尤其是在算术和代码相关任务中) 表现出改进。思维链提示 (Wei et al., 2022) 及其衍生技术，如零样本思维链 (Wang et al., 2023) 和思维程序 (Chen et al., 2022) (旨在将计算与推理

分开），已显著提升了多步骤推理。然而，尽管 CoT 可以提高在标准基准上的表现，我们的“打破链条”研究调查了一个关键的后续问题：当问题的表面结构被扰动时，即便核心逻辑得以保留，这种诱导的推理链是否仍然稳定有效。这个探索对于理解所激发出的推理的深度和可迁移性至关重要。

最近的调查和综合评论（Mondorf and Plank, 2024; Prasad et al., 2023; Golovneva et al., 2022）进一步探讨了 LLM 的推理行为，不仅限于简单的准确性指标，而是突出了一个普遍关注的问题：LLM 往往依赖于表面层次的相关性，而非真正的逻辑推理。例如，模型可能将某些关键词或措辞模式与特定的解答模板相关联。BREAK-THE-CHAIN 通过改变这些模式的方法直接测试这种依赖，例如讲故事、游戏化和领域转移，同时保持任务的基本语义完整性。这种对潜在表面提示的依赖性，即使在采用先进的提示技术时，也凸显了像我们这样的评估的必要性。因此，在这项工作中，我们旨在通过更细致的行为分析，而不仅仅是最终答案的正确性，来评估推理，重点关注在受控的语言和结构变化下的行为一致性。

2.2 用于代码生成的大型语言模型

大型语言模型在代码生成方面展现出了卓越的能力，通过将自然语言描述翻译为可执行代码来协助开发者。Jiang 等人进行了一个全面的调研（Jiang et al., 2024），对代码大型语言模型的最新进展进行分类，涵盖了数据集策划、基准测试评估和伦理影响等方面。评估大型语言模型在代码生成方面的性能仍然是一个重要的研究领域。传统方法主要依赖于基于执行的指标，如 pass@k，它评估生成的代码运行成功并通过测试用例的频率。更近期的基于大型语言模型的评估框架，如 CODEJUDGE（Tong and Zhang, 2024），利用语言模型自身来评估生成代码的语义正确性。

虽然这些方法很有价值，但诸如 pass@k 之类的指标主要评估标准问题形式上的功能正确性，而像 CODEJUDGE 这样的 LLM 评估器虽然评估语义正确性，但可能无法完全捕捉到模型在问题的表现本身被敌对但语义上扰动时的反应。我们的研究与稳健评估的目标一致，但通过关注在面对这种输入变化时导致代码生成的推理过程的稳定性来扩展它。BREAK-THE-CHAIN 专门评估这些敌对修改对代码生成中固有推理任务的影响，探测逻辑过程本身是否具备韧性。

2.3 推理 LLM 攻击

通过各种对抗攻击方法，LLM 的鲁棒性受到了仔细研究。先前的研究探讨了看似微小的扰动如何显著降低模型性能，例如拼写错误（Gan et al., 2024）或某些类型的对抗性提示（Wu et al., 2023）。例如，GSM-Symbolic 基准（Mirzadeh et al., 2024）表明对数学问题结构进行轻微修改，例如更改数值或添加无关条款，可能导致准确性显著下降，这表明当面对偏离已学习模式的输入时，LLM 推理存在根本性的脆弱性。

对代码生成模型的对抗攻击也进行了研究。DeceptPrompt 框架（Wu et al., 2023）揭示了表面上看似正常的自然语言修改可以诱使 LLMs 生成不安全或错误的代码，从而暴露现实应用中的漏洞。尽管 DeceptPrompt 展示了导致不安全代码的漏洞，BREAK-THE-CHAIN 探讨了一组重在语义的干扰（例如，讲故事、游戏化、领域转移、分散限制）。这些设计并不是主要为了诱发不安全性，而是为了测试在熟悉的问题提示被改变时，正确代码生成所需的基础推理链的稳定性和忠实性。我们的研究考察了这些对抗性修改——从叙事重组到注入逻辑上无效的信息——如何影响推理模型。我们方法的重要区别在于强调干扰，虽然在结构或框架上是对抗性的，但旨在在语义上忠实于原问题的核心逻辑。这使我们能够隔离并识别由于问题的展示方式变化而不是对完全不同任务的误解导致的推理失败。

总之，先前的工作已经确立了大型语言模型（LLMs）在复杂推理和代码生成领域的潜力，但也一贯强调了它们的脆弱性以及对学习到的表面模式的潜在过度依赖。现有的评估方法和攻击策略为了解模型能力和漏洞提供了有价值的见解。然而，在系统性理解 LLMs 的代码生成推理链如何经受住改变问题呈现但不改变核心逻辑的语义基础扰动方面，仍然存在一个空白。”BREAK-THE-CHAIN” 通过引入一个新框架和一套专门设计的扰动技术来解决这一问题，来探测代码生成领域中推理健壮性的这些细微方面。

3 方法论

为了系统地评估语义控制的修改如何影响大语言模型的代码生成能力，我们设计了一个框架，通过一组自然语言转换来扰动现有的编码问题。我们的方法侧重于对问题进行保留意义或最小化偏差的重写，而不改变其基本逻辑或难度。

3.1 问题扰动框架

不同于之前仅专注于对抗攻击或链式推理的工作（Mu et al., 2025; Zhu et al., 2025），我们的框架引入了编码问题的语义对齐改写，这些改写保留了原始编码问题的结构，同时在语言、叙述或逻辑框架上有所变化。受关于符号噪声（Mirzadeh et al., 2024）、基于叙述的越狱（Shen et al., 2024; Song et al., 2025）和欺骗性提示操纵的近期研究的启发（Wu et al., 2023），我们提出了四种变换策略（或“攻击”），在逻辑保留的程度上有所不同——从完全等同的重构（如讲故事）到目标逆转（如否定）。每种策略通过传递给 LLaMA 3.1 8B-Instruct 的一致提示模板实例化，引导对原始问题的重写，同时明确保留关键组件：输入、输出、解释、示例和约束。我们设计了不同类型的测试，以检查语言模型在几个关键领域的推理能力。讲故事和游戏化测试模型从非正式、人本叙述中提取形式逻辑的能力，这是现实世界用户提示中常见的情景。领域转移通过测试模型的算法知识是否依赖于特定术语环境来评估核心泛化能力。分散注意力的约束和示例扰动直接挑战模型的注意机制及其对表面模式匹配与稳健逻辑推理的依赖。最后，否定变换作为真实逻辑操作的压力测试，评估模型能否逆转已学推理过程而不是仅仅检索熟悉的一个。这些变换共同覆盖了从看似合理的语言变体到直接逻辑挑战的范围。

在本小节中，我们将详细描述每种问题扰动方法。

Attack Type	Score
Storytelling	8.89
Gamification	8.01
Domain Shift	7.07
Example Perturbation	6.71
Distracting Constraints	3.82

Table 1: 不同攻击类型的保护分数

讲故事。 受到安全研究中的叙事性越狱启发（Shen et al., 2024; Song et al., 2025; Chan et al., 2025），这种方法将问题重新构建为一个故事或冒险。这个提示保留了所有关键组件，但引入了一个可能改变表层结构的叙事包装。我们使用这种转变来研究在算法背景下，讲故事是否促进或阻碍大型语言模型的推理。讲故事的提示模板明确指示模型“使介绍部分有趣且引人入胜”，但不改变技术部分，如示例、约束或解释，同时在提示中明确提到“保持原始意图和结构”。这种转变测试了大型语言模型是否从更自然、更易于人类理解的措辞中受益，同时仍然保留了问题语义（见图 2）。

游戏化 这种转变将任务包装成一个基于挑战的形式（例如，玩家或机器人解决任务），同时保持逻辑不变。类似于讲故事，它引入拟人化或主题背景，从呈现在游戏场景中的任务的指令微调设置中汲取灵感（Wu et al., 2023）。我们评估这种游戏化输入是否能够提高参与度或分散模型的正式推理能力（见图 ??）。

受 GSM-Symbolic 中的符号注入策略启发，

干扰约束。 ，该方法将无关但听起来合理的约束（例如，“输入是回文”）附加到问题中。虽然逻辑上无关紧要，但这些约束测试模型过滤掉无关语言噪声的能力。与经典对抗性示例不同，这些并不改变任务的正确性，但增加了输入的模糊性（见图 ??）。

该策略将一个上下文中的标记和术语替换为另一个上下文中的标记和术语（例如，将” 整数数组” 替换为” 每日收入”）或使用特定领域的等价术语替换（例如，将数字转换为电梯楼层、职位、楼层、天数）（参见图 ??）。

3.2 逻辑保持评分

为了量化每种变换引起的语义偏离程度，我们开发了一个逻辑保留评分。我们采用 Claude-3.7-Sonnet 模型作为自动评估器，以确保在所有 700 个扰动实例中实现一致且可扩展的评分。此方法减轻了潜在人类评估者的疲劳和主观性。评分依据详细的基于评分标准的提示（参见附录 ??，图 ??），指示模型在 1-10 的范围内进行保留评分，其中 10 表示完美的逻辑对齐，1 表示完全的语义转变。这些评分将在后续分析中用于理解语义漂移与性能退化之间的关联。

4 实验设置

Model	Easy	Medium	Hard	Overall
Gemini-2.5-Flash	100.0	98.0	76.5	95.0
Claude-3.7-Sonnet	100.0	92.0	64.7	90.0
DeepSeek-14B	97.0	92.0	58.8	88.0
Gemini-2.0-Flash	97.0	86.0	47.1	83.0
DeepSeek-7B	90.9	70.0	35.3	71.0
Qwen2.5-Coder	81.8	52.0	35.3	59.0
DeepSeek-Coder-33B	84.8	48.0	11.8	54.0
Claude-3-Haiku	51.5	32.0	41.2	40.0
LLaMA-3.1-8B-Instruct	36.4	12.0	5.9	19.0
DeepSeek-Coder-1.3B	36.4	8.0	5.9	17.0
Avg. Accuracy	77.6	59.9	38.3	67.6

Table 2: 在无攻击设置下，不同难度级别下模型的精度（%）。

我们使用 LiveCodeBench 数据集（Jain et al., 2024）进行实验，这是一个近期设计

Storytelling Prompt Template

Instruction: Rewrite the problem in a storytelling format, while preserving its logic.

Guidelines:

- 保持输入、输出、解释、示例和约束部分不变。
- 将问题框定为一个引人入胜的故事或编程叙述。

{ original_content }

No additional explanation needed. Just output the modified problem.

Figure 2: 故事叙述转换的提示模板。

Models	Distracting Constraints	Domain Shift	Example Perturbation	Gamification	Storytelling	Avg. Acc.
Gemini 2.5 Flash	95.48 % (+0.5 %)	89.81 % (-5.2 %)	93.09 % (-1.9 %)	96.93 % (+1.9 %)	97.37 % (+2.4 %)	94.94 %
Gemini 2.0 Flash	88.78 % (+5.8 %)	72.47 % -10.5 %	95.02 % (+12.0 %)	89.76 % (+6.8 %)	90.38 % (+7.4 %)	87.28 %
DeepSeek-14B	83.84 % (-4.2 %)	75.53 % (-12.5 %)	84.10 % (-3.9 %)	86.76 % (-1.2 %)	91.16 % (+3.2 %)	84.28 %
Qwen2.5-Coder	65.03 % (+6.0 %)	68.33 % (+9.3 %)	83.51 % (+24.5 %)	70.11 % (+11.1 %)	79.69 % (+20.7 %)	73.33 %
DeepSeek-7B	65.06 % (-5.9 %)	58.28 % (-12.7 %)	71.18 % (+0.2 %)	73.30 % (+2.3 %)	82.99 % (+12.0 %)	70.16 %
DeepSeek-Coder-33B	58.44 % (+4.4 %)	55.26 % (+1.3 %)	70.00 % (+16.0 %)	67.44 % (+13.4 %)	70.45 % (+16.5 %)	64.32 %
Claude-3.7-Sonnet	47.89 % (-42.1 %)	35.66 % (-54.3 %)	62.87 % (-27.1 %)	50.00 % (-40.0 %)	63.35 % (-26.6 %)	51.95 %
Claude-3-Haiku	41.04 % (+1.0 %)	37.98 % (-2.0 %)	60.12 % (+20.1 %)	45.14 % (+5.1 %)	57.69 % (+17.7 %)	48.39 %
LLaMA3-8B-Instruct	37.59 % (+18.6 %)	22.03 % (+3.0 %)	54.30 % (+35.3 %)	37.78 % (+18.8 %)	44.68 % (+25.7 %)	39.28 %
Avg. Acc.	64.57 %	57.70 %	75.24 %	68.80 %	75.64 %	–

Table 3: 攻击准确度（ % ）以及从干净表现的差值 Δ 。行按平均准确度排序。

的基准，用于评估在交互和以测试为驱动的环境下的大型语言模型的代码生成能力。在我们的研究中，我们从 LiveCodeBench 的代码生成部分精选了 100 个 LeetCode 问题子集，这些问题提供了丰富的问题描述和适合修改和评估的测试用例。为了评估模型性能，我们采用标准的 Pass@1 指标（Jain et al., 2024），该指标衡量模型首次生成的解决方案通过所有提供的测试用例的问题比例。

我们评估了涵盖通用推理和代码专用能力的黑盒 API 模型和开源模型。为了与之前的工作保持一致，我们包括了来自 LiveCodeBench 排行榜的模型 [2] 以及原本不属于该基准的强代码聚焦基线。

我们测试了四种因其推理能力而著称的专有黑箱大语言模型：Gemini-2.5-Flash-Preview、Gemini-2.0-Flash（DeepMind, 2024）、Claude-3.7-Sonnet 和 Claude-3-Haiku（Anthropic, 2024）。为了评估开源推理，我们引入了 DeepSeek-R1-Distill-Qwen-7B 和 -14B（Guo et al., 2025），这些是经过蒸馏的 DeepSeek-R1 模型，具有强大的推理能力，并原生支持推理令牌生成，有助于细粒度的追踪分析。对于代码专注的模型，我们评估了：Qwen2.5-Coder 和 DeepSeek-Coder-33B，这两个模型都针对代码生成进行了优化，但没有专门调整以用于推理。最后，我们包括了 LLaMA-3.1-8B-Instruct（Meta, 2024），这是一个通用的指令调整模型。尽管不是代码或推理专用，LLaMA-3.1 的表现具有竞争力，并支持推理时的 CoT 提示工程，这使其成为我们分析中的一个有价值的参考点。

在本节中，我们展示了对大语言模型在扰动下评估的总体和手动案例级别的结果。虽然整体准确性和推理得分提供了对模型稳健性的宏观视角，但对特定实例的仔细审查揭示了细微的模式。我们确定了在某些情况下扰动有助于模型更好地推理，而在其他情况下则引入困惑，突出了提示形式对代码生成的多样影响。

4.1 性能概览清单

我们首先在 100 个未修改版本的 LeetCode 风格问题（表 ?? ）上对所有模型进行基准测试。如预期所料，针对指令遵循和推理进行优化的模型，如 Gemini-2.5-Flash（整体得分 95.0 %）和 Claude-3.7-Sonnet（90.0 %），优于特定代码模型如 Qwen2.5-Coder（59.0 %）和 DeepSeek-Coder-33B（54.0 %）。总体而言，从简单（77.6 %）到困难（38.3 %）任务的性能急剧下降，反映了推理复杂性在泛化中的内在挑战。

4.2 扰动鲁棒性和增益

表 ?? 显示，语义扰动并没有均匀地降低性能。相反，几种扰动类型——尤其是讲故事和游戏

[2] https://livecodebench.github.io/leaderboard.html

Attack	Model	Easy Acc.	Medium Acc.	Hard Acc.
Storytelling	Gemini-2.5-Flash	93.9 % (-6.1 %)	96.0 % (-2.0 %)	88.2 % (+11.7 %)
	Gemini-2.0-Flash	90.9 % (-6.1 %)	82.0 % (-4.0 %)	52.9 % (+5.8 %)
	Claude-3.7-Sonnet	48.5 % (-51.5 %)	34.0 % (-58.0 %)	47.1 % (-17.6 %)
	Claude-3-Haiku	45.5 % (-6.0 %)	26.0 % -6.0 %	35.3 % (-5.9 %)
	DeepSeek-14B	93.9 % (-3.1 %)	82.0 % (-10.0 %)	52.9 % (-5.9 %)
	DeepSeek-7B	97.0 % (+6.1 %)	62.0 % (-8.0 %)	23.5 % -11.8 %
Domain Shift	Gemini-2.5-Flash	89.8 % (-10.2 %)	96.0 % (+0.0 %)	82.4 % (+5.9 %)
	Gemini-2.0-Flash	72.5 % (-24.5 %)	80.0 % (-6.0 %)	58.8 % (+11.7 %)
	Claude-3.7-Sonnet	35.7 % (-64.3 %)	24.0 % (-68.0 %)	11.8 % -52.9 %
	Claude-3-Haiku	30.3 % (-21.2 %)	18.0 % (-14.0 %)	5.9 % (-35.3 %)
	DeepSeek-14B	54.5 % (-42.5 %)	62.0 % (-30.0 %)	29.4 % (-29.4 %)
	DeepSeek-7B	45.5 % (-45.4 %)	38.0 % (-32.0 %)	17.6 % (-17.7 %)
Example Perturbation	Gemini-2.0-Flash	97.0 % (+0.0 %)	92.0 % (+6.0 %)	64.7 % (+17.6 %)
	Gemini-2.5-Flash	93.9 % (-6.1 %)	84.0 % (-14.0 %)	70.6 % -5.9 %
	Claude-3.7-Sonnet	51.5 % (-48.5 %)	34.0 % -58.0 %	23.5 % (-41.2 %)
	Claude-3-Haiku	45.5 % (-6.0 %)	28.0 % (-4.0 %)	35.3 % (-5.9 %)
	DeepSeek-14B	81.8 % (-15.2 %)	68.0 % (-24.0 %)	47.1 % (-11.7 %)
	DeepSeek-7B	72.7 % (-18.2 %)	42.0 % (-28.0 %)	35.3 % (+0.0 %)
Distracting Constraints	Gemini-2.5-Flash	84.8 % (-15.2 %)	96.0 % (-2.0 %)	82.4 % (+5.9 %)
	Gemini-2.0-Flash	81.8 % (-15.2 %)	80.0 % (-6.0 %)	58.8 % (+11.7 %)
	Claude-3.7-Sonnet	36.4 % (-63.6 %)	24.0 % (-68.0 %)	11.8 % (-52.9 %)
	Claude-3-Haiku	36.4 % (-15.1 %)	12.0 % (-20.0 %)	17.6 % -23.6 %
	DeepSeek-14B	78.8 % (-18.2 %)	70.0 % (-22.0 %)	41.2 % (-17.6 %)
	DeepSeek-7B	57.6 % (-33.3 %)	36.0 % (-34.0 %)	29.4 % (-5.9 %)
Gamification	Gemini-2.5-Flash	100.0 % (+0.0 %)	92.0 % (-6.0 %)	88.2 % (+11.7 %)
	Gemini-2.0-Flash	97.0 % (+0.0 %)	74.0 % (-12.0 %)	58.8 % (+11.7 %)
	Claude-3.7-Sonnet	48.5 % (-51.5 %)	30.0 % (-62.0 %)	23.5 % -41.2 %
	Claude-3-Haiku	51.5 % (+0.0 %)	34.0 % (+2.0 %)	35.3 % (-5.9 %)
	DeepSeek-14B	84.8 % (-12.2 %)	64.0 % (-28.0 %)	52.9 % (-5.9 %)
	DeepSeek-7B	75.8 % (-15.1 %)	46.0 % (-24.0 %)	29.4 % (-5.9 %)

Table 4: 在各种攻击下，注重推理的大型语言模型的准确性（%），按难度级别划分。

化——始终能提高性能，特别是对那些调整为人类对齐推理的模型来说。

值得注意的是，Gemini-2.0-Flash 在干净准确率上排名第四，在四种扰动类型中表现优于其原始基线，在例子扰动上提高了至多 +12.0 %。同样地，Qwen2.5-Coder 和 LLaMA-3.1，虽然并非专长于推理，均在叙事风格或模式打破重写下表现出显著的性能提升（分别为 +24.5 % 和 +35.3 %）。基于这些发现，我们注意到某些模型从更自然或去中心化的问题表述中获益。这表明僵化、极简的问题陈述可能实际上未能充分利用大型语言模型的能力。这一点在我们对 Gemini-2.5-Flash 输出的人工检查中得到了进一步佐证，在 11 个干扰约束攻击的案例中，添加了约束的扰动版本生成了正确的解决方案，而原始版本则没有。尽管这些约束被标记为“干扰性”，但它们可能使隐含假设变得显性，提高了清晰度。这与 Xu et al. (2024) 的发现一致，其表明在提示中使用结构化语境可以提高生成准确性。这个发现挑战了此前认为自然语言冗长或格式偏离总是有害的假设。反之，我们观察到语义更加丰富的提示可以激发更好的程序合成。

尽管存在一般的趋势，但并非所有模型对语义变化的抵抗力相同。Claude-3.7-Sonnet 在低保留转化下的准确度大幅下降，在中等难度的分散约束任务中，准确度下降幅度高达 -68.0%。相比之下，Gemini-2.5-Flash 和 DeepSeek-14B 显得非常稳定，在所有任务中的偏差仅为 5%。通过这些实验结果，我们展示了一种新的评估轴：语义扰动的鲁棒性——定义为在逻辑保留重写中的性能标准差。该轴捕捉了模型在同一任务的不同良性形式下保持性能的能力。我们认为这种鲁棒性与纯净的准确性同等重要，特别是在真实世界应用中，用户查询的语言变化多端且措辞不够完美。这一现象在诸如 Claude-3.7-Sonnet 等模型中可见，当受到扰动的提示生成代码时，往往比原始的更简单且不够完整。似乎该模型依赖于原始提示中的表面模式来检索结构化代码，而扰动则破坏了这种检索行为。正如我们观察到的，这种扰动可能会增加令牌级别的熵和方差，使模型信心降低，且更容易出错。

在图 3 中，我们将平均模型准确性与我们的逻辑保留分数（表 ?? ）进行对比绘图。虽然是非线性的，正相关性出现了：讲故事（分数：

Attack	Model	Easy Acc.	Medium Acc.	Hard Acc.
Storytelling	Qwen2.5-Coder	84.8 % (+3.0 %)	54.0 % (+2.0 %)	35.3 % (+0.0 %)
	DeepSeek-Coder-33B	81.8 % (-3.0 %)	34.0 % (-14.0 %)	23.5 % (+11.7 %)
	LLaMA3-8B-Instruct	45.5 % (+9.1 %)	4.0 % (-8.0 %)	29.4 % (+23.5 %)
Example Perturbation	Qwen2.5-Coder	84.8 % (+3.0 %)	66.0 % (+14.0 %)	41.2 % (+5.9 %)
	LLaMA3-8B-Instruct	45.5 % (+9.1 %)	28.0 % (+16.0 %)	11.8 % (+5.9 %)
	DeepSeek-Coder-33B	72.7 % (-12.1 %)	44.0 % (-4.0 %)	17.6 % (+5.8 %)
Distracting Constraints	Qwen2.5-Coder	57.6 % (-24.2 %)	42.0 % (-10.0 %)	17.6 % (-17.7 %)
	DeepSeek-Coder-33B	54.5 % (-30.3 %)	26.0 % -22.0 %	29.4 % (+17.6 %)
	LLaMA3-8B-Instruct	30.3 % (-6.1 %)	8.0 % (-4.0 %)	17.6 % (+11.7 %)
Gamification	Qwen2.5-Coder	75.8 % (-6.0 %)	52.0 % (+0.0 %)	23.5 % -11.8 %
	DeepSeek-Coder-33B	69.7 % (-15.1 %)	50.0 % (+2.0 %)	35.3 % (+23.5 %)
	LLaMA3-8B-Instruct	42.4 % (+6.0 %)	20.0 % (+8.0 %)	23.5 % (+17.6 %)
Domain Shift	Qwen2.5-Coder	64.3 % (-17.5 %)	41.2 % (-10.8 %)	20.0 % (-15.3 %)
	DeepSeek-Coder-33B	48.5 % (-36.3 %)	26.0 % (-22.0 %)	17.6 % (+5.8 %)
	LLaMA3-8B-Instruct	15.2 % (-21.2 %)	2.0 % (-10.0 %)	11.8 % (+5.9 %)

Table 5: 在各种攻击下代码生成模型的准确性 (%)。

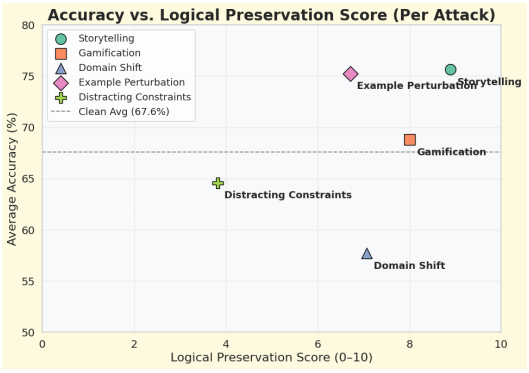

Figure 3: 每种攻击类型的准确性与逻辑保留分数。尽管存在较大的改写，但讲故事和示例扰动提高了性能，而干扰性约束则显著降低了准确性。

8.89）和游戏化（8.01）会导致性能提高或保持稳定。分散注意力的约束（3.82）和否定目标（1.83）则会持续降低性能。然而，我们注意到性能不会随着逻辑偏移成比例地下降。一些逻辑上保存的重写能提高准确性，而结构上简单的编辑（如分散注意力的约束）会显著降低其性能。这表明认知对齐——而不仅仅是逻辑对齐——主导了大型语言模型的成功。大型语言模型对语义干扰物比对叙事抽象更敏感。

如表 ?? 所示，模型在所有攻击中通常在简单问题上保留高精度，但即使在高保持重写的情况下，在困难任务下也会崩溃。这证实了推理复杂性，而不仅仅是输入措辞，会导致模型崩溃。然而，叙述方法显著提升了 Gemini-2.5 (+11.7 %) 和 LLaMA3.1 (+23.5 %) 在困难问题上的表现，这表明人性化的重新表述可能有助于在挑战条件下进行更深入的推理。

4.3 通过提示清晰度产生的错误

为了阐明为什么某些扰动有助于而其他扰动却造成妨碍，我们借鉴了最近的研究成果，这些研究将模型不确定性与错误率联系起来。Cai et al. (2024) 观察到，在 LLM 中，错误更可能发生在高熵和方差区域的符号处，即模型对下一步不确定的地方。当我们扰动提示时，尤其是加入模糊或无关的约束时，生成的提示可能会使模型的注意力更加分散，增加不确定性，导致生成错误的结果。

这突显了扰动的双重性质：当它们在不增加结构的情况下破坏预期模式时，会降低性能；但当它们详细说明或搭建问题背景时，可能会像上面表 ?? 中某些模型那样提高推理能力。未来的模型可能会受益于不仅针对正确答案进行调整，还针对具有不确定性意识的推理路径进行调整。

4.4 评论驱动的推理行为

在分析 Claude-3.7-Sonnet 在游戏化提示变体下的表现时，出现了一个显著的行为模式。虽然 Claude-3.7 在干净条件下表现出色（整体准确率为 90.0 %；表格 ?? ），但在游戏化情况下其准确率急剧下降了 40.0 %（表格 ?? ）。对模型输出（表格 ?? ）的人工检查揭示了行为的一致性偏移：Claude 在未扰动提示中的代码回答常常包含内联注释——解释控制流逻辑、边缘情况处理或特定变量的目的。相比之下，在游戏化提示措辞下的回答则缺乏这样的解释性注释。

自然语言注释的缺失似乎与代码质量下降有关。在干净设置中正确实现的逻辑在扰动下往往变得脆弱或不一致，表明内部推理一致性降低。这个趋势与 Dou et al. 的研究结果一致，

Original Prompt Output	Perturbed Prompt Output	Original Result	Perturbed Result

class Solution:
def countSeniors(self, details: List[str]) − > int:
# 计数器用于 60 岁以上的乘客 	
count = 0
for passenger in details:
年龄保存在第 11 和第 12 个字符中
age = int(passenger[11:13])
# 检查年龄是否严格大于 60 	
if *age* > 60 :
count += 1
return count
	class Solution:
	def countSeniors(self, details: List[str]) − > int:
	count = 0
	for record in details:
	age = int(record[12:14])
	if *age* > 60 :
	count += 1
	return count
		[True, True]	[False]

Table 6: 在提示扰动下的评论驱动推理行为：比较 Claude-3.7-Sonnet 在纯净（原始）和游戏化（扰动）提示下的代码响应和结果准确性。

(Dou et al., 2024) 他们表明，在大语言模型生成的代码中，注释频率越高，推理能力越强，错误越少。他们的结果表明，在代码合成期间鼓励生成注释可以提高推理能力——即使不进行模型的微调也是如此。

我们的研究结果扩展了这一见解，提出评论生成不仅是内部推理的反映，还作为模型鲁棒性的中介因素。当 Claude-3.7 在提示扰动下省略评论时，其性能显著下降——尽管底层任务语义保持不变。这支持了一个假设，即基于评论的推理在保持相同任务不同语言表达形式的性能方面可能起到稳定作用。

4.5 深入解析领域转移

对评估日志的详细分析揭示了不同的行为模式。首先，Llama-3.1 的稳健性似乎来源于其能够从表面叙述中抽象出潜在的逻辑任务的能力。其推理经常涉及用新的领域术语重新表述问题，然后应用正确的、通用的算法，有效地处理语义转换（见附录 C.1）。此外，Qwen2.5-Coder 的性能提升表明，转向更符合常见编程任务的领域或使用更具体的术语（例如，用"employee_salaries" 代替抽象的"nums"）可能增强了其理解或激发了更相关的学习模式。它始终采用新的术语，同时正确地实现核心逻辑（见附录 C.1）。另一方面，Claude 的准确性大幅下降突显出在面对不熟悉的主题背景时的显著脆弱性。多次情况中，Claude 的推理（输出）能够在新领域中正确地重新表述目标，但其生成的解决方案代码却极其不完整，仅解决了问题的一个更简单版本，或者未能实现核心保留逻辑。例如，在 minIncrementOperations 任务（原本是根据滑动窗口条件使数组"美观"，转变为基于类似窗口的任务稳定性）中，Claude 正确地理解并重新表述了新领域术语中的复杂窗口目标。然而，其随后生成的代码仅实现了一个更简单且错误的逐元素检查，完全遗漏了所需的窗口逻辑（见附录 C.2）。这表明尽管它可以处理表层语义变化，但映射到正确复杂算法解决方案的能力变得不可靠。DeepSeek-14B 虽然没有像 Claude 那样全面失败，但仍然显示出一些困难。其错误通常涉及正确的一般算法思想，但在新领域中的执行有缺陷，比如误解新术语或在调整条件时出错，这表明在将熟悉的逻辑准确地翻译到新的语义环境中存在挑战（见附录 C.1）。

为了探究语义鲁棒性的边界，我们进行了一项消融研究，重点关注否定目标（困难）和否定目标（软性）转化。这些重写旨在测试模型是否能够适应目标的反转或微妙的反转，同时保持原始问题的格式和结构元素。

4.6 方法

否定目标（严格） 这种方法完全颠倒了核心任务（例如，"最大化" → "最小化" 或 "包括" → "排除"）。转换提示的一部分要求 LLM："用相反的目标改写……确保例子和输出相应更新：（见图 ?? ）。虽然输入、输出、解释和约束被保留，但解决方案逻辑必须改变，要求模型从新的问题意图重新计算。根据 Claude-3.7，保留分数记录为 1.83。

否定目标（软性）。 这种变体采用了较温和的语义反转（例如，要求 "非递减" 而不是 "递增"），而没有完全颠覆任务。指令明确指出："不要从根本上改变问题的任务……尽量少地调整例子"（见图 ?? ）。这一设计测试模型是否对微妙的语言调整具有鲁棒性，这些调整可以改变逻辑但保持可解性。根据 Claude-3.7，保留分数记录为 2.56，略高于强否定。

4.7 结果

表格 ?? 总结了基于否定的扰动的影响。强否定具有高度破坏性，导致在所有难度级别上平均准确率下降超过 50 %。例如，Gemini-2.5-Flash 的准确率从 95.0 % 下降到 42.6 % ，而 Claude-3.7 从 90.0 % 降至 15.0 % 。即便是

遵循指令的模型也难以适应，因为需要推理反转，而不仅仅是语言适应。相比之下，软否定表现出更多变化，一些模型如 LLaMA-3.1 的准确率从 19.0 % 提高到 29.4 % 。然而，其他模型如 DeepSeek-14B 和 Claude-3.7 仍然遭遇大幅下降，显示出即使微小的逻辑变化也能破坏推理。

在软否定下，Llama-3.1-8B-Instruct 的改进准确性有时与测试用例与修改后的文本目标正确对齐时的真实适应相关（见附录 A.1）。然而，一个普遍的问题是测试用例不一致：尽管 Llama 对修改后的文本进行了正确的逻辑适应，但仍然未通过仍期望原始问题答案的测试（例如，针对 sumOfSquares ，附录 A.2 ）。相反，像 DeepSeek-14B 这样的模型有时通过忽视修改而“通过”，因为它们的测试用例也保持不变。其他模型的失败模式包括部分适应，其中只实现了部分否定逻辑（例如，Gemini-2.5-Flash 在 matrixSum 上，附录 A.3 ），以及原型覆盖，其中模型在矛盾的修改指令下恢复到已知的问题解决方案（例如，Gemini-2.5-Flash 在 maximumOr 上，附录 A.4）。特别是 DeepSeek-14B，一贯忽视修改。虽然 Llama-3.1 表现出对文本变化更好的语义忠实度，但其整体准确率的提高是一个由真实适应和与测试用例一致性的复杂互动所影响的微妙结果。真正的鲁棒性评估依赖于精心对齐的问题提示和测试评估。

否定目标（难）分析。 “否定目标（困难）”的扰动导致所有模型的任务几乎完全失败，报告的准确率（0% – 17% ）无法反映在处理反向目标时的真实能力，因为广泛存在的测试用例不匹配。观察到的“成功”通常出现在模型对原始问题进行求解，但测试用例未更新的情况下（例如，Qwen2.5-Coder 在 count_seniors ，附录 B.1 ；Gemini-2.5-Flash 在 min_extra_char ，附录 B.4 ）。相反，正确实现简单否定逻辑的模型经常在这些不匹配的测试中失败（例如，Claude, DeepSeek, Gemini 在 count_seniors ，附录 B.2 ）。没有模型能够始终如一地表现出推理反转的能力。相反的，常见的行为包括默认为原始问题的原型、代码不稳定（如 DeepSeek-14B 在复杂否定上的表现，附录 B.3 ）以及即便在承认文本变化的情况下仍无法正确实现反向逻辑（例如，matrixSum ，附录 B.5 ）。因此，困难否定暴露了当前大型语言模型在执行真实目标和逻辑反转方面的深刻局限性，强调了需要进行严格对齐的评估的重要性。

总之，基于否定的重写揭示了大型语言模型推理中的一个关键盲点：即便在所有结构信息都被保留的情况下，仍然无法重新解释或反转逻辑目标。此外，模型在否定目标下的崩溃表明任务框架能够覆盖指令调优，并且模型难以从逆逻辑中重新计算。最后，软否定揭示了词汇理解的脆弱性：人类认为微不足道的小语义扰动可能导致推理路径上的连锁失败。

我们工作的一个限制是，我们仅评估了 LiveCodeBench 中 LeetCode 风格问题的一个子集——具体来说是 235 个问题中的 100 个。此外，我们的研究没有涵盖 LiveCodeBench 中可用的其他任务类型，例如代码执行、测试输出预测和自我修复。虽然 LiveCodeBench 包含一套多样化的问题源，除了 LeetCode，还来自于 AtCoder 和 Codeforces 等平台，但我们的评估仅限于 LeetCode 子集。做出这个决定主要是由于计算资源有限，因为随着模型规模的增大，特别是开源模型，推理时间会显著增加。为了确保不同模型之间的公平比较，我们保持了一致的评估组。

尽管存在这些限制，我们通过生成和评估 700 个扰动实例（100 个干净问题 × 7 种扰动类型）进行大规模鲁棒性分析，使我们能够系统地评估模型在不同逻辑修改下的行为。

另一个局限是排除了 GPT 系列模型（例如 GPT-4），根据 LiveCodeBench 排行榜，这些模型是代码生成表现最好的专有模型之一。由于获取限制，我们无法在评估中包含它们，因此在这些扰动下它们的稳健性仍然是未来研究的一个开放问题。

5 结论

在这项工作中，我们提出了 Break-The-Chain ，这是一个系统化的框架，用于在代码生成任务中评估大语言模型（LLMs）在对抗性提示下的推理鲁棒性。通过引入语义上合理但逻辑上具有挑战性的扰动，例如叙事、游戏化、干扰性约束和否定目标，我们揭示了当前最先进模型中的关键漏洞。通过针对九个 LLM 和 700 个扰动实例的全面实验，我们发现尽管一些扰动出乎意料地改善了模型性能，但其他扰动，尤其是涉及微妙语义漂移的那些，导致了显著的推理失败。

我们的分析引入了一个新的评估轴，语义扰动鲁棒性，其量化了模型在逻辑上等价但语言上多样的输入中进行泛化的能力。这些发现强调，高的清洁准确性并不意味着稳健的推理能力，并突出显示了需要进行更符合认知的模型训练。

References

Anthropic (2024). Claude 3.5 sonnet model card addendum. https://www.anthropic.com/news/

claude-3-5-sonnet. Accessed: 2025-05-12.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. (2021). Program synthesis with large language models. arXiv preprint arXiv:2108.07732.

Bai, Y., Kadavath, S., Kundu, S., et al. (2023). Constitutional ai: Harmlessness from ai feedback. arXiv preprint arXiv:2306.04761.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33.

Cai, Y., Donhauser, J., Salemi, A., McDowell, P., Tian, Y., and Ghassemi, M. (2024). Attention misalignment: Modeling errors in language models through entropy and variance. arXiv preprint arXiv:2503.21961.

Chan, Y. S., Ri, N., Xiao, Y., and Ghassemi, M. (2025). Speak easy: Eliciting harmful jailbreaks from llms with simple interactions. arXiv preprint arXiv:2502.04322.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. (2022). Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. arXiv preprint arXiv:2211.12588.

Chowdhery, A., Narang, S., Devlin, J., et al. (2022). Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311.

DeepMind, G. (2024). Start building with gemini 1.5 and gemini 2.5. https://developers.googleblog.com/en/start-building-with-gemini-25-flash/. Accessed: 2025-05-12.

Dou, S., Jia, H., Wu, S., Zheng, H., Zhou, W., Wu, M., Chai, M., Fan, J., Huang, C., Tao, Y., Liu, Y., Zhou, E., Zhang, M., Zhou, Y., Wu, Y., Zheng, R., Wen, M., Weng, R., Wang, J., Cai, X., Gui, T., Qiu, X., Zhang, Q., and Huang, X. (2024). What's wrong with your code generated by large language models? an extensive study. arXiv preprint arXiv:2407.06153.

Gan, E., Zhao, Y., Cheng, L., Mao, Y., Goyal, A., Kawaguchi, K., Kan, M.-Y., and Shieh, M. (2024). Reasoning robustness of llms to adversarial typographical errors. arXiv preprint arXiv:2411.05345.

Golovneva, O., Chen, M., Poff, S., Corredor, M., Zettlemoyer, L., Fazel-Zarandi, M., and Celikyilmaz, A. (2022). Roscoe: A suite of metrics for scoring step-by-step reasoning. arXiv preprint arXiv:2212.07919.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. (2024). Livecodebench: Holistic and contamination free evaluation of large language models for code. arXiv preprint arXiv:2403.07974.

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. (2024). A survey on large language models for code generation. arXiv preprint arXiv:2406.00515v2.

Meta (2024). Introducing llama 3.1: Our most capable models to date. Meta-AI-blog.

Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. (2024). Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. arXiv preprint arXiv:2410.05229.

Mondorf, P. and Plank, B. (2024). Beyond accuracy: Evaluating the reasoning behavior of large language models-a survey. corr, abs/2404.01869, 2024. doi: 10.48550. arXiv preprint ARXIV.2404.01869.

Mu, W., Xu, L., Pei, S., Mi, L., and Zhou, H. (2025). Evaluate-and-purify: Fortifying code language models against adversarial attacks using llm-as-a-judge. https://arxiv.org/abs/2504.19730.

OpenAI (2023). Gpt-4 technical report. https://openai.com/research/gpt-4. Accessed: 2025-05-12.

Prasad, A., Saha, S., Zhou, X., and Bansal, M. (2023). Receval: Evaluating reasoning chains via correctness and informativeness. arXiv preprint arXiv:2304.10703.

Shen, X., Wu, Y., Backes, M., and Zhang, Y. (2024). Voice jailbreak attacks against gpt-4o. arXiv preprint arXiv:2405.19103.

Song, X., Xie, Z., Huai, S., Kong, J., and Luo, J. (2025). Dagger behind smile: Fool llms with a happy ending story. arXiv preprint arXiv:2501.13115.

Tong, W. and Zhang, T. (2024). Codejudge: Evaluating code generation with large language models. arXiv preprint arXiv:2410.02184.

Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., and Lim, E.-P. (2023). Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. arXiv preprint arXiv:2305.04091.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837.

Wu, F., Liu, X., and Xiao, C. (2023). Deceptprompt: Exploiting llm-driven code generation via adversarial natural language instructions. arXiv preprint arXiv:2312.04730.

Xu, Y., Yaghini, M., Ogutu, K., Sun, Z., Lin, X. V., and Xiong, C. (2024). Prompting under control: Structured prompts guide language models better. arXiv preprint arXiv:2412.20545.

Zhu, Y., Li, G., Jiang, X., Li, J., Mei, H., Jin, Z., and Dong, Y. (2025). Uncertainty-guided chain-of-thought for code generation with llms. arXiv preprint arXiv:2503.15341.

A 否定目标（软性）分析的详细示例

A.1 示例 A.1: countBeautifulPairs —驼羊的成功适应

原始问题目标（总结）： 计算数对 (nums[i], nums[j]) 的数量，其中 i < j 满足 nums[i] 的第一位数字和 nums[j] 的最后一位数字互质（即，最大公约数为 1）。
统计这样一对数：其中 nums[i] 的第一个数字和 nums[j] 的最后一个数字不是互质数（即，最大公约数大于 1）。

输入实例： nums = [30, 72, 9, 96, 82]

Llama-3.1-8B-Instruct 性能：(result: True)

- 推理片段："这个问题要求我们计算非美丽对的数量……如果 nums[i] 的第一个数字和 nums[j] 的最后一个数字不是互质的，则该对是非美丽的。这意味着它们的最大公约数（GCD）大于 1。"
- 相关的 solution_code 逻辑：

```
if gcd(first_digit_i, last_digit_j) > 1:
    count += 1
```

- 结果详情： { 'output': '3', 'expected': '3' }
- 分析：Llama 正确地实现了修改后的逻辑，并与预期结果相符。

Claude-3.7-十四行诗表现: (result: False)

- 推理/代码逻辑：实现了逻辑以计数 GCD == 1（原始问题）。输出 7 。
- 结果详情： { 'output': '7', 'expected': '0' }
- 分析：未能适应被否定的任务，并未通过其自身的测试。

DeepSeek-14B 性能: (result: False)

- 与 Claude 相同：使用了原始问题逻辑。输出：7 ，预期：0 。

双子座-2.5-闪电预览性能: (result: False)

- 推理：正确地适应了否定逻辑。
- 输出：3 ，但期望：0 。
- 分析：调整后的逻辑与测试用例的期望不符。

比较总结： Llama 成功地与逻辑和期望对齐。Gemini 部分成功。Claude 和 DeepSeek 未能适应。

A.2 示例 A.2: sumOfSquares —羊驼适应，测试用例期望原始版本

原始问题目标（总结）： 求和元素 nums[i] 的平方，其中以 1 为基准的索引 i 可被 n 整除（即 n % i == 0 ）。

修改后的问题目标（软否定）： 求那些索引 i 不能整除 n 的元素的平方的和（即，n % i != 0 ）。

输入实例： nums = [1, 2, 3, 4] （n = 4 ）

- 原始正确结果：$1^2 + 2^2 + 4^2 = 21$
- 修改后的正确结果：$3^2 = 9$

Llama-3.1-8B-Instruct 绩效: （result: False ）

- 相关 solution_code 逻辑：

```
sum(num ** 2 for i, num in enumerate(nums, start=1) if n
```

- 输出：9 ，预期：21
- 分析：Llama 正确地遵循了修改后的逻辑，但由于测试用例期待原始问题的输出而失败。

DeepSeek-14B 性能：(result: True)

- 相关 solution_code 逻辑：

```
if n
    ans += nums[i-1]**2
```

- 输出：21 ，预期：21
- 分析：DeepSeek 忽略了修改后的指令并解决了原始问题。由于测试用例期望值不匹配，它通过了测试。

比较总结： Llama 忠实地应用了取反逻辑，但由于预期不正确而未通过测试。DeepSeek 忽略了指令的变化，却通过了测试，这凸显了对齐评估流程的必要性。

A.3 示例 A.3: matrixSum —由 Gemini 进行的部分调整

从每一行中移除最大元素，取这些元素中的最大值并将其添加到得分中。

修改后的问题目标（否定）： 对于每一行，移除最小的元素，从中取最小值，并从分数中减去它。
Gemini-2.5-Flash-Preview 性能：（ result: False ）

- 相关 solution_code 逻辑：

```
for row in nums:
    row.sort()
for j in range(num_cols):
    removed_this_round = []
    for i in range(num_rows):
        removed_this_round.append(nums[i][num_cols - 1 - j])

    min_val = min(removed_this_round)
    score -= min_val
```

- 分析：Gemini 正确地应用了减法并从最大聚合切换到最小聚合，但依然从每一行中移除了最大值——未能完全适应问题的逻辑。

A.4 例 A.4: maximumOr —双子座的原型覆盖

在给定 nums 和 k 操作（每个操作允许一个数字乘以 2）的情况下，最大化所有元素的按位或。

修改后的问题目标（否定）： 在经过 k 操作后最小化按位或值。
Gemini-2.5-Flash-Preview 性能：（result: True ）

- 推理行为: 尽管指令要求最小化，Gemini 求解了最大 OR。输出与原始测试用例相匹配，保持不变。
- 输入：nums = [12, 9] ， k = 1
- 输出：30 ， 期望：30
- 分析：Gemini 覆盖了被否定的目标，可能是由于对已知问题模式的过拟合，并且通过了测试案例，因为测试案例仍然与最初的公式保持一致。

比较摘要： 示例 A.3 展示了部分适应性：Gemini 识别出了转换（例如，减法、最小值），但在行级逻辑上失败了。在示例 A.4 中，模型完全回到了原始目标，这反映了记忆化原型对提示指令的支配。

B 否定目标（难度）分析的详细示例

B.1 示例 B.1: count_seniors —测试不对齐 & 错误成功

原始问题目标（总结）： 统计严格大于 60 岁的乘客。

修改后的问题目标（硬否定）： 严格统计年龄在 60 岁或以下的乘客（即，$age \leq 60$ ）。

输入实例： details = ["7868190130M7522", "5303914400F9211", "9273338290F4010"]

- 原始正确输出（年龄 > 60）：2（对应年龄 75, 92）
- 修改后的正确输出（年龄 ≤ 60）：1（针对年龄 40）

Qwen2.5-Coder 性能：（result: [True, True] ）

- 相关 solution_code 逻辑：

```
if age > 60: # Original problem logic
    count += 1
```

- 结果详情： { 'output': '2', 'expected': '2' }
- 分析：Qwen 忽略了否定的提示（"60 或以下"），执行了原始逻辑（"大于 60"），并且通过了。这强烈表明其测试用例期望的是原始问题的答案。

Claude-3.7-Sonnet / DeepSeek-14B / Gemini-2.5-Flash 性能：(result: [False])

- 分析：参见附录 B.2 。这些模型实现了正确的负逻辑，但失败了。

对比总结： Qwen " 成功" 地通过应对一个不对齐的测试解决了原始任务。其他模型正确地适应了否定的文本，但在相同的不对齐测试中失败了。

B.2 示例 B.2: count_seniors —正确的否定未通过未对齐测试

与附录 B.1 相同。

- 原始正确输出：2
- 修改后的正确输出：1

Claude-3.7-十四行诗性能：（result: [False] ）

- 相关的 solution_code 逻辑：

```
if age <= 60: # Correct negated logic
    count += 1
```

- 结果详情： { 'output': '1', 'expected': '2', 'error_message': 'Wrong Answer' }
- 分析：Claude 正确实现了修改后提示所要求的逻辑（"60 或更少"），但失败是因为测试案例期望的是与原始提示（"大于 60"）相对应的输出。

DeepSeek-14B & 双子-2.5-闪存性能：（result: [False] ）

- 逻辑 & 结果详情：与 Claude 相同的结果。两者都实现了 age <= 60 ，输出了 1，但测试预期为 2。

比较总结： 多个模型忠实地适应了文本中的简单否定，但由于测试案例期望原始问题的行为而被错误地惩罚。

B.3 例 B.3：find_missing_and_repeated_values —深度搜索在复杂否定上的代码不稳定性

原始问题目标（总结）： 找出一个从网格元素派生的范围内出现两次的数字和缺失的数字。

修改后的目标（强硬否定 - 示意性）： 可以被反转以找到出现一次且存在的数字，或其他复杂的逻辑逆转。

DeepSeek-14B 性能：(result: [-4])

- 相关 solution_code 片段（说明）：通常涉及复杂的理解或逻辑，试图处理否定的任务。

```
# Code failed during execution
# Example error: 'NoneType' object is not iterable
```

- 结果细节： { 'error': 'TypeError("NoneType object is not iterable")', 'error_code': -4 }
- 分析：DeepSeek 试图推理复杂的逆向逻辑导致了运行时错误。这表明，在困难问题上逆向目标超出了模型的能力，导致代码不稳定。

B.4 例 B.4: min_extra_char —双子的错误成功通过原始逻辑

在使用字典中的单词对字符串 s 进行最优分割后，找到多余字符的最小数量。

修改后的问题目标（严格否定）： 找到最多的额外字符数。

输入实例： s = "leetscode" , dictionary = ["leet", "code", "leetcode"]

- 原始正确输出（最少额外）：1（使用 "leetcode"）
- 修改后的正确输出（最大额外）：9（不使用单词）

Gemini-2.5-Flash 性能：(result: [True, True])

- 推理片段（output ）：描述了使用动态规划方法寻找最少的额外字符数。
- 相关 solution_code 逻辑：

```
# dp[i] will store the minimum number of extra chars
# ... logic calculates minimum ...
return dp[n]
```

- 结果详情：测试案例通过。执行时间表明运行成功。
- 分析：Gemini 忽略了被否定的目标（"最大化"），而是解决了原始目标（"最小化"）。成功表明测试用例与被否定的提示不一致。

比较总结： Gemini 通过恢复到已知的原始问题原型而取得成功，这可能是由于一个错误配置的测试用例仍然评估了原始目标。

B.5 例子 B.5: matrixSum —模型在复杂否定上的混淆

原始问题目标（总结）： 逐行迭代地移除最大的数，找到其中的最大值，并将其加入分数中。

修改后的问题目标(硬否定)： 迭代地从每一行中移除最小的数字, 找出其中最低的一个, 然后从分数中减去它。

DeepSeek-14B 性能：(result: [False])

- 推理片段："所以过程是：1. 对于每一行，去掉最小的元素。2. 在这些元素中找到最小值。3. 减去这个最小值……等等，不对……"
- 相关 solution_code 逻辑：

```
rows = [deque(row) for row in nums]
sum_min = 0
while any(rows):
    # Find and accumulate min_val
    sum_min += min_val
return -sum_min
```

- 结果详情：标记为 "错误答案"。
- 分析：DeepSeek 试图采用一种通过否定的逻辑形式，但在推理和实现上表现出困惑。

Claude, Gemini, Qwen 性能表现：(result: [False])

- 分析：所有模型都失败了。Gemini 显示出即时混乱；Qwen 退回到简化逻辑；Claude 在目标倒置上遇到了困难。

比较总结： 此任务的多步骤、逻辑反转性质导致所有模型失败。这揭示了一个在处理需要非平凡推理调整的程序性否定时的共同弱点。

C 领域偏移分析的详细示例

C.1 示例 C.1: 问题 - ” 计算和小于目标的数对”（简单难度）

给定一个从 0 开始索引的整数数组 ‘nums‘和一个整数 ‘target‘，返回满足 ‘0 <= i < j < nums.length‘且 ‘nums[i] + nums[j] < target‘的对数 ‘(i,j)‘。

给定一个员工工资列表 ‘employee_salaries‘ 和一个目标 ‘bonus_threshold‘。你的任务是计算员工对 (i, j) 的数量，其中员工 ‘i‘ 排在员工 ‘j‘ 前面，并且他们工资的总和 ‘employee_salaries[i] + employee_salaries[j]‘ 严格小于 ‘bonus_threshold‘。(措辞改编自 Llama 的日志)。迭代配对并检查其总和是否达到阈值的核心逻辑被保留。

输入实例（来自原始问题，适用于这里的所有模型）： ‘nums = [-1,1,2,3,1]‘，‘target = 2‘。

本实例的正确输出（原始逻辑和移位逻辑）： 3。

Llama-3.1-8B-Instruct 性能: (result: True)

- 推理片段 (output) : ” 问题要求我们统计雇员对 (i, j) 的个数，其中 i < j，并且他们的总薪水 (‘employee_salaries[i] + employee_salaries[j]‘) 严格小于给定的 ‘bonus_threshold‘... 我们可以遍历所有可能的雇员对 (i, j) 使得 i < j。对于每一对，我们计算他们薪水的总和。如果总和小于 ‘bonus_threshold‘，我们就增加一个计数器。”

- 相关 solution

  _code 片段:

```
class Solution:
    def countPairs(self, employee_salaries: List[int], bonus_threshold: int) -> int:
        n = len(employee_salaries)
        count = 0
        for i in range(n):
            for j in range(i + 1, n):
                if employee_salaries[i] + employee_salaries[j] < bonus_threshold:
                count += 1
        return count
```

- 结果详细信息 (metadata_output): ‘ ’ 输出’: ’3’, ’ 输入’: ’[[-1, 1, 2, 3, 1], 2]’, ’ 期望值’: ’3’, ... ‘

- 分析: Llama 成功理解了领域转移，正确使用了新的术语，并实现了正确的配对计数逻辑，与预期输出一致。

Qwen2.5-Coder 性能: (result: True)

- 推理片段 (output): “问题要求我们计算员工对 (i, j) 的数量，使得 ‘employee_salaries[i] + employee_salaries[j] < bonus_threshold‘。”

- 相关的 solution_code 片段: 正确实现使用新的域变量名称，类似于 Llama。

- 结果详情 (metadata_output): ‘ ’ 输出’: ’3’, ’ 输入’: ’[[-1, 1, 2, 3, 1], 2]’, ’ 期望’: ’3’, ... ‘

- 分析: Qwen 也成功地适应了这一点，支持了将转换到具体编程场景可以很好处理的想法。

DeepSeek-14B 性能: (result: False)

- 推理片段 (output): 最小化推理，通常直接输出代码。

- 相关的 solution_code 片段 (此问题的失败示例):

```
# Might correctly implement the logic but use original names, or make subtle errors
class Solution:
    def countPairs(self, nums: List[int], target: int) -> int: #Note: uses 'nums','target'
        n = len(nums)
        count = 0
        # ... (correct loop structure) ...
        # if nums[i] + nums[j] < target: count += 1
        return count # For this instance, output was 0
```

示例 C.1 的比较总结： Llama 和 Qwen 成功推广了这个配对计数任务。DeepSeek 的失败表明在新的语义环境中准确地重新应用算法存在困难。Claude 在其他问题上的典型失败（如在 C.2 中所见）显示出更严重的中断。

C.2 示例 C.2: 问题 - minIncrementOperations - Claude-3.7-十四行诗的失败通过过度简化

给定 'nums'和 'k'，找到最小的递增操作次数，使得对于每个长度为 3 的子数组，至少有一个元素 '>= k'。(这是一个动态规划问题)。

修改后的问题目标（Claude 的领域偏移示例）： " 你正在管理项目任务。'task_priorities'是一个初始优先级分数的数组，'min_threshold'是一个整数。操作包括将某个任务的优先级增加 1。如果对任意连续三个任务的序列中，至少有一个任务的优先级分数大于等于 'min_threshold'，则项目进度表是' 稳定' 的。返回使进度表稳定所需的最少优先级增加次数。"

输入实例： 'nums = [2,3,0,0,2]'（变为 'task_priorities'），'k = 4'（变为 'min_threshold'）。

此实例的正确输出（原始和移位逻辑）： 3。

Claude-3.7-十四行诗表现：(result: [False])

- 推理片段 (output): " 问题要求我们找出使得一个计划 "稳定" 的最小 '优先级提升'（操作）次数。一个计划是由数组 'task_priorities'表示的，如果每个连续 3 个任务的序列中至少有一个任务的优先级分数大于或等于 'min_threshold'，则该计划是稳定的..."
- 相关 solution_code 片段：

```
class Solution:
    def minIncrementOperations(self, task_priorities: List[int], min_threshold: int)->int:
        n = len(task_priorities)
        ans = 0
        # Incorrectly makes EACH element >= threshold, not the windowed condition
        for i in range(n):
            if task_priorities[i] < min_threshold:
                ans += min_threshold - task_priorities[i]
        return ans
```

D 否定目标和否定目标（软）结果

Attack	Model	Easy Acc.	Medium Acc.	Hard Acc.	Overall
Negation Objective (Hard)	DeepSeek-7B	3.0 % (-87.9 %)	2.0 % (-68.0 %)	0.0 % -35.3 %	14.0 % (-57.0
	DeepSeek-14B	3.0 % (-94.0 %)	0.0 % (-92.0 %)	0.0 % (-58.8 %)	5.7 % (-82.3
	Qwen2.5-Coder	33.3 % (-48.5 %)	20.0 % (-32.0 %)	23.5 % (-11.8 %)	47.2 % (-11.8
	LLaMA3-8B-Instruct	3.0 % (-33.4 %)	0.0 % (-12.0 %)	0.0 % -5.9 %	8.3 % -10.
	DeepSeek-Coder-33B	18.2 % -66.6 %	22.0 % -26.0 %	5.9 % -5.9 %	36.9 % (-17.1
	Claude-3-Haiku	12.1 % (-39.4 %)	4.0 % (-28.0 %)	5.9 % (-35.3 %)	21.2 % (-18.8
	Claude-3.7-Sonnet	6.1 % (-93.9 %)	2.0 % (-90.0 %)	5.9 % (-58.8 %)	15.0 % (−75.0
	Gemini-2.5-Flash	27.3 % (-72.7 %)	18.0 % (-80.0 %)	23.5 % (-53.0 %)	42.6 % (-52.4
	Gemini-2.0-Flash	18.2 % (-78.8 %)	26.0 % (-60.0 %)	11.8 % (-35.3 %)	43.2 % (-39.8
Negation Objective (Soft)	DeepSeek-7B	3.0 % (-87.9 %)	0.0 % (-70.0 %)	0.0 % -35.3 %	5.7 % (-65.3
	DeepSeek-14B	0.0 % (-97.0 %)	0.0 % (-92.0 %)	0.0 % (-58.8 %)	1.1 % (-86.9
	Qwen2.5-Coder	36.4 % (-45.4 %)	22.0 % (-30.0 %)	5.9 % -29.4 %	46.9 % (-12.1
	LLaMA3-8B-Instruct	21.2 % (-15.2 %)	6.0 % (-6.0 %)	5.9 % (+0.0 %)	29.4 % (+10.4
	DeepSeek-Coder-33B	0.0 % (-84.8 %)	25.0 % (-23.0 %)	0.0 % (-11.8 %)	37.5 % (-16.5
	Claude-3-Haiku	15.2 % (-36.3 %)	0.0 % (-32.0 %)	11.8 % (-29.4 %)	22.5 % (-17.5
	Claude-3.7-Sonnet	12.1 % (-87.9 %)	2.0 % (-90.0 %)	5.9 % (-58.8 %)	25.4 % (-64.6
	Gemini-2.5-Flash	21.2 % (-78.8 %)	24.0 % (-74.0 %)	11.8 % (-64.7 %)	41.5 % (-53.5
	Gemini-2.0-Flash	15.2 % (-81.8 %)	30.0 % (-56.0 %)	5.9 % (-41.2 %)	40.6 % (-42.4

Table 7: 对于被否定目标和否定目标（软性）扰动的问题集的准确性（%）。

E 修改的提示模板

F 保留分数提示模板

Gamification Prompt Template

Instruction: Rewrite the problem as a challenge involving agents or players.
Guidelines:

* 保留所有技术组件（输入、输出、说明、约束）。
* 添加一个类似游戏的叙述层（例如，机器人导航，解谜）。

Prompted Task: Embed the problem into a goal-driven challenge structure.
{ original_content }
No additional explanation needed.

Figure 4: 用于游戏化转换的提示模板。

Example Perturbation Prompt Template

Instruction: Modify only the examples to make them confusing for LLMs.
Guidelines:

* 保持输入/输出逻辑正确。
* 打乱值，插入噪声（例如，交换标签，极端情况的放置）。

Prompted Task: Make the examples harder to pattern-match while maintaining validity.
{ original_content }
No additional explanation needed.

Figure 5: 示例扰动转换的提示模板。

Distracting Constraints Prompt Template

Instruction: Add irrelevant but realistic constraints to the problem.
Guidelines:

* 不要改变可解性。
* 插入边缘情况、无意义的术语或会分散模型注意力的限制。

Prompted Task: Inject non-functional complexity into the problem description.
{ original_content }
No additional explanation needed.

Figure 6: 分心约束变换的提示模板。

Domain Shift Prompt Template

Instruction: Shift the problem into a different but equivalent domain.
Guidelines:

* 保留逻辑结构和约束。
* 替换名词和背景（例如，整数 → 工资，数组 → 电梯楼层）。

Prompted Task: Change the framing without affecting logic.
{ original_content }
No additional explanation needed.

Figure 7: 域转换的提示模板。

Hard Negation Objective Prompt Template

Instruction: Invert the objective of the coding task.
Examples:

* 最大化 → 最小化
* 递增 → 非递增
* 包括 → 排除

Prompted Task: Rewrite the problem with the opposite intent and updated examples.
{ original_content }
No additional explanation needed.

Figure 8: 用于硬否定目标转换的提示模板。

Soft Negation Prompt Template

Instruction: Apply a light semantic reversal without changing the task's core logic.
Examples:

* 不减而不是增加
* 非最小而不是最小

Guidelines:

* 保持输入/输出、示例和难度。
* 仅调整任务描述和最小示例逻辑。

Prompted Task: Rewrite with slight negation while keeping format identical.
{ original_content }
No additional explanation needed.

Figure 9: 软否定转换的提示模板。

Logical Preservation Evaluation Prompt Template

System Prompt: You are a meticulous evaluator tasked with assessing the logical preservation of coding problems.
You will be given two versions of a coding problem: the Original Problem and the Modified (Perturbed) Problem . Your goal is to evaluate whether the Modified Problem faithfully preserves:

* 基本任务
* 输入/输出规范，
* 逻辑结构
* 以及原始问题的所有关键细节。

You must judge strictly according to the rubric provided. Be critical but fair: minor rewordings are acceptable, but changes to task meaning, core logic, or critical information are not. You must output your evaluation in a structured format.

User Prompt: You will now receive two problems: an Original Problem and a Modified Problem.
{ original_problem }
{ modified_problem }
Please answer the following evaluation questions carefully:

1. 任务一致性：修改后的问题是否要求用户解决与原始问题相同的基本任务？
2. 输入/输出保留：输入格式、输出格式和约束是否在逻辑上相同？
3. 逻辑完整性：问题的解决路径和必要的推理是否得到保留？
4. 关键信息：所有重要的信息（约束、条件、定义）是否都在未失真的情况下被保留？
5. 总体保存评分（0–10）：
   * 10 = 完美保留，只进行表面措辞更改。
   * 8–9 = 极小的无害变化; 核心任务完全保持完好。
   * 6–7 = 一些小的逻辑转换；仍然大部分可以以相似的方式解决。
   * 4–5 = 主要的逻辑转换；重要遗漏或令人困惑的修改。
   * 1–3 = 核心任务被更改；引入了实质性的混淆。
   * 0 = 任务完全改变或对抗性破坏。
6. 推理：写一个简要但详细的解释来证明这个分数的合理性。如果存在具体的差异或问题，请指出。

Format your output exactly like this:
Task Consistency: [Yes/No]
Input/Output Preservation: [Yes/No]
Logical Integrity: [Yes/No]
Critical Information: [Yes/No]

Preservation Score: [0–10]

Reasoning:
- [Brief, clear explanation highlighting matching parts or critical deviations]

Figure 10: 用于评估原始和修改后的编码问题之间逻辑保持的提示模板。