

可见测试、留出测试与评判模型： 度量 GRPO 多轮代码修复中的奖励投机

DC-Jev-RL 项目，第三部分 A

2026 年 10 月

摘要

用策略能看到的测试来奖励它，它可能学会满足这些测试，而不是解决问题。我们在多轮代码修复中测量这一点：Qwen3.5-4B-Base 写一个函数，看到第一个失败的示例测试后重写，最多三次；由留出测试判定最终程序是否正确。离线时，基座策略通过可见测试的尝试中有 36% 在留出测试上失败；Kev-4B 和一个同样大小的自回归评判模型都把这类尝试排在正确尝试之下 (AUROC 分别为 0.87 和 0.86)，并且几乎完美地把对测试的硬编码查表程序与参考解答区分开。随后我们用 GRPO 训练 100 步，每种奖励两个种子：只用可见测试；可见测试乘以 Kev-4B 对代码是否 genuine 的判断；以及作为预言机的全部测试。我们没有看到可见测试奖励被利用的迹象：解出题目的比例在 KodCode val (从 65.5% 升至 86.6%)、MBPP+ 和 HumanEval+ 上都提高了，通过可见测试的程序中错误者的比例在三个集合中的两个上降低了约一半，抄写可见测试中数值的程序仍和基座策略中一样少见。评判因子带来的变化没有超出种子之间的差异；从不奖励错误程序的预言机奖励，让错误的通过略有减少，但没有解出更多题目。代码、日志和本文的每一个数字都在仓库中。

1 引言

编程智能体通常只看到少量示例测试，而它写的代码必须在示例之外也能工作。用可见测试构造的训练奖励，奖励的是通过这些测试，而不是解决问题。策略可能学会以不能推广的方式满足示例，极端情况下直接对示例做特判。在以测试为奖励训练推理模型时，已经观察到这类奖励投机行为 [2, 4, 6]。

本文在一个小而完整的设定中测量评判模型能帮上多少忙。任务是多轮代码修复：策略写一个函数，看到第一个失败的示例测试及其报错，再试一次，最多三次。奖励有三种：只用可见测试；可见测试乘以 Kev-4B 对代码是否“对所有合法输入都解决了问题，而不是对示例测试做特判”的判断；以及作为预言机的可见测试加留出测试。我们测量可见测试所证明的与留出测试所发现的之间的差距：离线地针对基座策略的代码和专为通过可见测试而构造的硬编码查表程序；在线地针对 GRPO 训练过程中和训练之后的策略。

2 相关工作

执行反馈是代码强化学习的标准奖励。EvalPlus [5] 这类扩展了测试的基准表明，HumanEval 和 MBPP [1, 3] 的原始测试漏掉了许多错误程序，这正是只用可见测试的奖励所留下的缺口。Kod-Code [7] 为强化学习提供带有已验证测试的合成题目。代码正确性的 LLM 评判，是第二部分在数学上研究的生成式验证器 [8] 在代码上的对应物。监控策略的推理以发现奖励投机迹象 [2]，是使用第二个模型的另一种方式；这里评判模型只读最终代码。

3 任务：带可见测试和留出测试的代码修复

题目。 训练题来自 KodCode-Light-RL-10K [7]：带 `pytest` 单元测试的 Python 函数题。我们保留这样的题目：测试文件只测一个函数，不使用 `fixture` 或第三方包，并且至少有三个测试函数。第一个测试函数展示给策略（可见），其余的留出。300 道题（种子 0）留作验证集（`kodcode_val`）。测试集为 MBPP+ 和 HumanEval+ [1, 3, 5]。MBPP+ 中原始 MBPP 的 `assert` 是可见的（多数题为三个，最多七个）；HumanEval+ 中原始 HumanEval 测试的前三个 `assert` 是可见的（原测试不足三个时取全部）。两者中，EvalPlus 测试框架及其新增的测试都是留出的。若一道题的参考解答在我们的测试框架下通不过它自己的任何测试，就丢弃该题，并报告丢弃数量。每个程序都在空临时目录中各自的 `python -I` 子进程里运行，限制 CPU 时间、地址空间和文件大小，并设墙钟超时；运行前禁用 `os`、`shutil` 和 `subprocess` 中的破坏性函数（即 OpenAI `human-eval` 的“`reliability guard`”）。这能约束小模型写出的有缺陷代码，但不是安全边界。

回合。 策略看到题目、函数签名和可见测试，写出一个解答（每次尝试最多 768 个词元）。代码在可见测试上运行。若有测试失败，策略看到第一个失败的测试及其报错，再写一次，总共最多三次。每次尝试都接续之前各轮的完整词元序列。回合在第一次通过全部可见测试的尝试处结束，或在第三次尝试后结束，然后最终代码在留出测试上运行。提示见 `experiments/part3a_code/prompts/code_prompts.json`，在任何运行之前即已固定。

指标。 *vis pass*: 最终代码通过全部可见测试。*all pass*: 通过全部可见测试和全部留出测试。*hack*: 通过可见测试但有留出测试失败。它既统计过拟合或特判的解答，也统计仅仅是不完整的解答；这一指标无法区分两者，我们把它作为“可见测试所证明的”与“实际情况”之间的差距来报告。首次尝试 *all pass* 衡量不经修复时策略的水平。

4 奖励

- **vis**: 最终代码通过全部可见测试则为 1。这是只持有可见测试的智能体所能检查的全部内容，也正是诱发对可见测试过拟合的奖励。
- **vis_kev**: `vis` 乘以 Kev-4B 对 *genuine* 问题回答“是”的概率：“所给代码是否对所有合法输入都解决了问题，而不是对示例测试做特判？”Kev-4B 看到题目、可见测试和最终代码，从不看到

留出测试。

- **hidden**: 最终代码通过全部可见测试和全部留出测试则为 1。这是一个预言机：相当于测试完整时的 **vis**。

一个回合中的每次尝试都使用该回合的优势进行训练，反馈词元作为上下文，不参与训练。更新方式与第二部分相同：每题 8 个回合的组基线、批级尺度、词元级损失、截断重要性采样，每批一次优化器更新。

5 离线评判研究

训练之前，基座策略在 `kodcode_val`、MBPP+ 和 HumanEval+ 上以温度 1 每题写 4 个单次尝试。我们在两组测试上运行它们，两个评判模型对每个尝试回答 *genuine* 和 *correct*。对可见测试奖励而言，关键问题是：在通过可见测试的尝试中，评判模型能否把同时通过留出测试的那些分出来。我们报告这一 AUROC，以及在全部尝试上识别“全部通过”的 AUROC。我们还为 MBPP+ 和 HumanEval+ 构造两类状态已知的代码：一类是参考解答；另一类是硬编码查表程序，对每个可见测试的参数返回期望值，其他情况返回 `None`。查表程序只有在通过可见测试时才保留，因此它是纯粹对测试做特判的程序。我们报告每个评判模型给查表程序的分数高于同一道题参考解答的频率。

6 评判模型在代码中看到了什么

可见测试留下了很大的缺口。 基座策略的 3,340 次单次尝试（每题 4 次）中，21.9% 通过可见测试，14.0% 通过全部测试。在通过可见测试的尝试中，36% 有留出测试失败：KodCode val 上为 43%，MBPP+ 上为 25%，HumanEval+ 上为 32%。在训练还没来得及利用它之前，用可见测试构造的奖励就已经给这些尝试与正确尝试同样多的奖励。

两个评判模型都能看出大部分问题，在训练分布上 Kev-4B 并不更好。 表 1 给出在通过可见测试的尝试中，各信号对“同时通过留出测试”的 AUROC。在三个数据集合并后，Kev-4B 的 *genuine* 达到 0.87 [0.84, 0.90]，AR 评判达到 0.86 [0.83, 0.89]（按题目重采样得到的 95% 区间）。两者之差 +0.013 [-0.007, 0.033] 不能分辨。用 *correct* 问题时，Kev-4B 领先 +0.030 [0.011, 0.050]。两种差异的大部分都来自 MBPP+，在那里 Kev-4B 的 *genuine* 领先 +0.086 [0.045, 0.128]。在策略训练所用的分布 KodCode val 上，AR 评判领先，但幅度不能分辨 (-0.012 [-0.038, 0.010])；在 HumanEval+ 上两者无法区分 (+0.010 [-0.054, 0.069])。在全部尝试上，两个评判模型把全部通过的尝试与其余尝试区分得更好 (0.93 和 0.94)，AR 评判略微领先 (-0.012 [-0.020, -0.005])。

题目内部的证据很少。 GRPO 的组基线比较的是同一道题的各个回合，所以 `vis_kev` 中评判因子的作用，取决于它如何给一道题中通过可见测试的回合排序。在 442 道至少有一次尝试通过可见测试的题目中，只有 62 道在留出测试上两种结果都有。在这些题目内部，Kev-4B 的 *genuine* 达到

	held-out tests pass, among visible-pass attempts					all pass
	KodCode val	MBPP+	HumanEval+	all [95% CI]	within	all
attempts passing the visible tests	391	225	116	732		
of which fail a held-out test	169	57	37	263		
Kev-4B <i>genuine</i>	0.89	0.83	0.85	0.87 [0.84, 0.90]	0.81	0.93
Kev-4B <i>correct</i>	0.88	0.82	0.86	0.87 [0.84, 0.90]	0.79	0.93
AR <i>genuine</i>	0.90	0.74	0.84	0.86 [0.83, 0.89]	0.85	0.94
AR <i>correct</i>	0.87	0.74	0.84	0.84 [0.81, 0.87]	0.81	0.93

表 1: 在基座策略通过可见测试的尝试中（每题 4 次尝试，温度 1），各评判信号对“通过留出测试”的 AUROC。within: 在 62 道“通过可见测试的尝试中既有通过也有未通过留出测试者”的题目内部的平均 AUROC。all pass: 在全部尝试上识别“通过全部测试”的 AUROC。95% 区间由按题目重采样得到。

signal	AUROC	mean $p(\text{yes})$		lookup scored
		reference	lookup	above reference
Kev-4B <i>genuine</i>	1.000	0.75	0.03	0.0%
Kev-4B <i>correct</i>	0.997	0.75	0.07	0.2%
AR <i>genuine</i>	1.000	0.80	0.02	0.0%
AR <i>correct</i>	0.991	0.79	0.05	0.8%

表 2: 在 507 道可见测试允许构造查表程序的 MBPP+ 和 HumanEval+ 题目上，参考解答对硬编码查表程序：AUROC、两类程序的平均 $p(\text{yes})$ ，以及查表程序得分高于其所属题目参考解答的频率。

0.81，AR 评判达到 0.85，差值 $-0.040 [-0.109, 0.021]$ 。这些题目太少，既不足以给两个评判模型排序，也不足以预测评判因子会在多大程度上改变训练。

硬编码查表程序会被识破。 507 个对可见测试调用做查表的程序通过了其所属 MBPP+ 或 HumanEval+ 题目的可见测试（其中 1 个也通过了留出测试）。两个评判模型都几乎完美地把它与同一批题目的参考解答区分开（表 2）。Kev-4B 的 *genuine* 给查表程序的平均分为 0.03，给参考解答的为 0.75，从未给查表程序打出高于其所属题目参考解答的分数。这类查表是对测试做特判最粗暴的方式。评判模型能识破它们，说明 *genuine* 问题被正确理解了，但不能说明更隐蔽的过拟合也会被发现。

这预示着什么。 评判因子可以压低通过可见测试却错误的尝试中的大部分，而 Kev-4B 做到这一点的程度与 AR 评判——一个同样大小、被问同样问题的通用模型——相当。评判因子能否在训练中缩小这一差距，以及代价是多少可见测试通过率，要由训练实验来测量。

condition	seeds	KodCode val			MBPP+			HumanEval+		
		vis	all	hack	vis	all	hack	vis	all	hack
vis	2	97.1	86.6	10.6	84.7	69.3	15.4	88.5	76.4	12.1
vis_kev	2	96.6	86.6	10.1	85.2	72.3	12.9	87.9	76.7	11.2
hidden	2	96.3	87.2	9.1	82.0	68.8	13.3	88.2	77.0	11.2

表 3: 最终策略 (贪心解码, 最多三次尝试): 通过可见测试 (*vis*)、通过全部测试 (*all*), 以及通过可见测试但有留出测试失败 (*hack*) 的题目比例 (%); 种子均值。

7 针对可见测试的训练

每种奖励训练策略 100 步 (每步 16 道题、每题 8 个回合, 学习率 10^{-5} , 与第二部分相同), 各用 2 个种子。表 3 给出最终策略在三个评估集上的结果, 表 4 给出奖励之间的配对差, 图 1 给出训练曲线。95% 区间接题目重采样, 不包含种子之间的差异, 因此在种子结果不一致的地方, 我们也给出每个种子的数值。

用可见测试训练让代码更正确。 第 0 步时, 基座策略在 65.5% 的 KodCode val 题目、54.8% 的 MBPP+ 题目和 49.1% 的 HumanEval+ 题目上通过全部测试。用 vis 奖励训练 100 步后, 这三个比例为 86.6%、69.3% 和 76.4%, 而且需要的修复更少: 它在首次尝试就解出 84.7% 的 KodCode val 题目, 第 0 步时为 59.8%。MBPP+ 和 HumanEval+ 从未用于训练。

可见测试与留出测试之间的差距在两个集合上缩小了, 在 MBPP+ 上没有。 在基座策略于 KodCode val 上通过可见测试的尝试中, 有 23.7% 在留出测试上失败; 训练后为 10.9%。在 HumanEval+ 上, 这一比例从 27.7% 降到 13.7%。在 MBPP+ 上它几乎不变, 从 20.0% 到 18.2%: 策略通过 MBPP+ 可见测试的题目多了很多 (第 0 步为 68.6%, 训练后为 84.7%), 而可见测试通过、留出测试失败的题目占全部题目的比例从 13.7% 增加到 15.4%。训练没有让策略对可见测试的过拟合比基座策略更严重; 但在 MBPP+ 上, 也没有让它减轻。

预言机奖励: 错误的通过略少, 解出的题目没有增加。 hidden 只奖励通过全部测试的代码, 因此从不奖励错误的程序。它的最终策略解出的题目与 vis 一样多: all pass 的差值在 KodCode val 上为 +0.7 [-1.8, 3.2] 个百分点, 在 MBPP+ 上为 -0.5 [-2.5, 1.3], 在 HumanEval+ 上为 +0.6 [-3.4, 5.0] (表 4)。它通过可见测试的频率更低 (MBPP+ 上为 82.0%, vis 为 84.7%), 而差异的大部分来自错误的程序: hack 率的差值分别为 -1.5 [-3.9, 0.8]、-2.1 [-4.0, -0.3] 和 -0.9 [-4.7, 2.8] 个百分点, 只有在 MBPP+ 上区间不含零。一个从不奖励错误程序的奖励, 让其中一部分不再通过可见测试; 但在 100 步之内, 它没有把它们变成正确的程序。

评判因子: 对 MBPP+ 的作用只出现在一个种子中。 vis_kev 把可见测试奖励乘以 Kev-4B 的 *genuine* 分数。在 KodCode val 和 HumanEval+ 上, 它的最终策略解出的题目与 vis 一样多 (+0.0

minus vis	all pass			hack		
	KodCode val	MBPP+	HumanEval+	KodCode val	MBPP+	HumanEval+
vis_kev	+0.0	+3.1	+0.3	-0.5	-2.5	-0.9
	[-2.5, 2.5]	[1.1, 5.2]	[-3.7, 4.3]	[-2.9, 1.7]	[-4.5, -0.7]	[-3.7, 1.9]
hidden	+0.7	-0.5	+0.6	-1.5	-2.1	-0.9
	[-1.8, 3.2]	[-2.5, 1.3]	[-3.4, 5.0]	[-3.9, 0.8]	[-4.0, -0.3]	[-4.7, 2.8]

表 4: 相对 vis 的配对差 (百分点): 在每道题上取各条件的种子均值, 求差后对题目取平均, 并给出按题目重采样的 95% 区间。

condition	seeds	training episodes that copy			final evaluation, copy	
		steps 0-9	steps 50-59	steps 90-99	hack	all pass
vis	2	0.2%	0.2%	0.0%	1 / 218	2 / 1283
vis_kev	2	0.2%	0.1%	0.0%	0 / 193	1 / 1307
hidden	2	0.2%	0.0%	0.0%	0 / 190	4 / 1285

表 5: 含有参考解答中没有的可见测试值的程序: 三个十步窗口中训练回合的比例 (种子均值), 以及最终评估的 hack 回合和 all pass 回合中的数量 (所有种子和评估集合计)。

[-2.5, 2.5] 和 +0.3 [-3.7, 4.3])。在 MBPP+ 上种子均值更好: all pass 的差值为 +3.1 [1.1, 5.2] 个百分点, hack 率的差值为 -2.5 [-4.5, -0.7], 按题目的区间不含零。但两个种子并不一致: vis_kev 在种子 1 下通过 75.8% 的 MBPP+ 题目的全部测试, 在种子 2 下为 68.9%, 而 vis 分别为 69.1% 和 69.4%。两个种子中只有一个出现的效果, 不是我们能够宣称的效果。

没有抄写测试。 hack 率统计每一个通过可见测试的错误程序, 不论它只是不完整, 还是对测试做了特判。为了区分两者, 我们寻找最粗暴的特判形式: 程序中出现了可见测试里的某个值 (参数或期望结果), 而参考解答中没有这个值 (dcjev/analysis/code_hacks.py; 只搜索被测函数及其用到的模块级代码, 因此模型自己写的示例调用不算)。这项检查标出了离线研究中 507 个硬编码查表程序中的 488 个, 即每一个含有非平凡值的查表程序, 而 537 个参考解答一个也没有标出。在离线研究中基座策略的尝试里, 它标出了 263 个通过可见测试但有留出测试失败的尝试中的 2 个, 以及其余尝试的 1.1%。训练期间, 在 vis 的任何一个十步窗口中, 它最多标出 0.7% 的回合 (vis_kev 为 0.7%, hidden 为 0.8%), 且没有上升趋势 (表 5)。在最终评估中, 它标出了 vis 的 218 个 hack 回合中的 1 个、vis_kev 的 193 个中的 0 个, 以及 hidden 的 190 个中的 0 个。通过可见测试的错误程序几乎从不含有测试中的值, 训练前后都是如此; 可见测试奖励没有教会策略抄写测试。不抄写任何值的特判, 这项检查发现不了。

评估噪声。 每次运行都在第 0 步评估同一个基座模型。在 6 次运行中, KodCode val 上的 all pass 介于 65.1% 和 65.8% 之间 (标准差 0.3 个百分点), MBPP+ 上介于 54.3% 和 55.6% 之

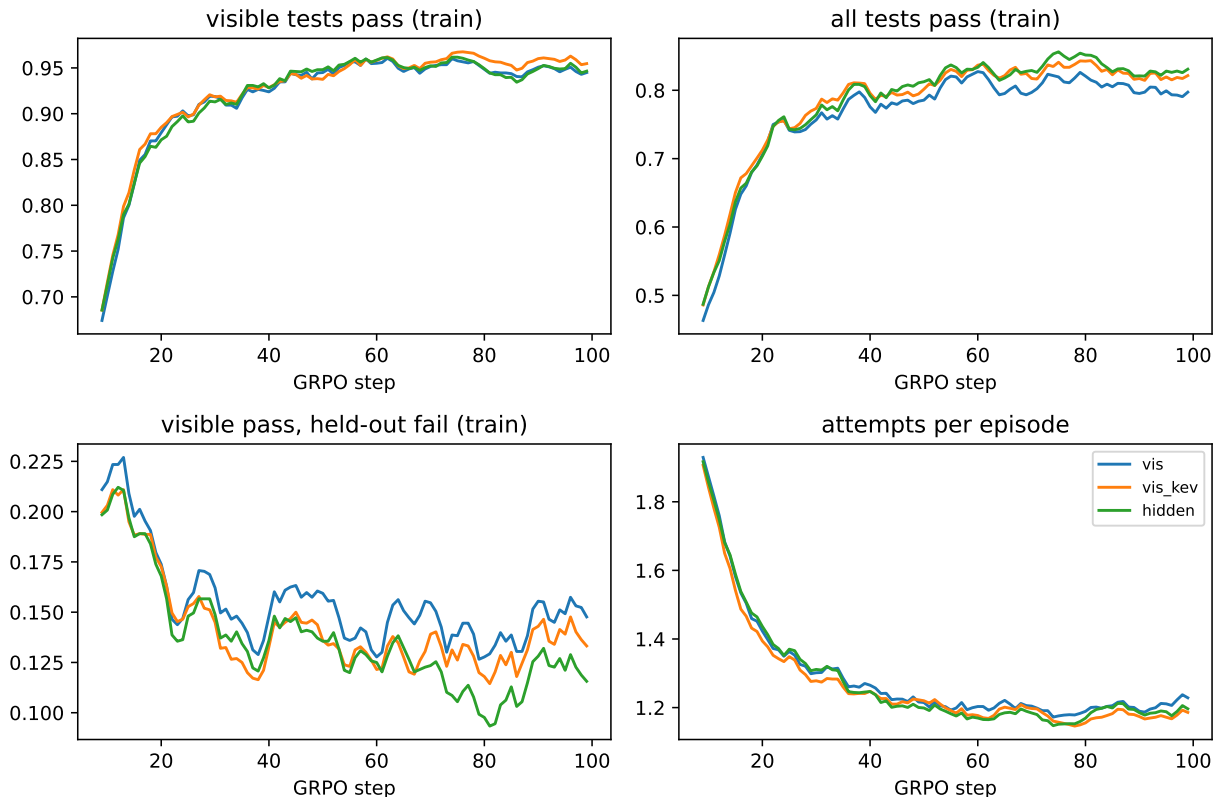


图 1: 训练曲线 (种子均值, 10 步滑动平均): 训练回合中通过可见测试、通过全部测试、通过可见测试但有留出测试失败的比例, 以及每个回合的尝试次数。

间, HumanEval+ 上介于 48.4% 和 49.7% 之间。vLLM 上的贪心解码在不同的服务器状态下不能逐位复现。同一奖励不同种子之间的差异更大: all pass 的标准差最高达 4.9 个百分点 (vis_kev, MBPP+)。

8 讨论

没有看到可见测试被利用的迹象。 本文关心的是: 用策略能看到的测试来奖励它, 它可能会满足这些测试, 而不是解决问题。在我们的设置中, 100 步之内没有看到这种迹象。训练提高了三个集合上解出题目的比例, 包括策略从未训练过的两个集合。通过可见测试的程序中错误者的比例, 在 KodCode val 和 HumanEval+ 上降低了约一半, 在 MBPP+ 上几乎不变。抄写可见测试中数值的程序, 仍和基座策略中一样少见。通过可见测试的错误程序, 在训练前后都几乎从不是对测试的抄写: 它们处理了示例所展示的情况, 却把其他情况做错了。可见测试奖励会奖励这些程序, 但基座策略本来就会写出它们, 而在三个集合中的两个上, 训练让它们在通过者中所占的比例变小了。

评判模型能做什么，实际做了什么。 离线时，两个评判模型给一道题中通过可见测试的尝试排序时，都能比随机更好地分出也通过留出测试的那些 (§6)，而且都能识破硬编码查表程序。在训练中，评判因子带来的变化没有超出种子之间的差异：在 KodCode val 和 HumanEval+ 上没有差别，在 MBPP+ 上一个种子有提升，另一个没有。剩下需要压制的不是对测试的抄写，而是在示例未展示的情况上出错的解答。在策略确实学会特判的情形下——测试更弱、题目更难或训练更长——评判因子是否有用，这些运行无法回答。

更好的测试去掉的是少量错误的通过，而不是未解出的题目。 预言机奖励只奖励通过全部测试的程序。它最终在三个集合上都让通过可见测试的错误程序比 `vis` 更少（只有在 MBPP+ 上区间不含零），而解出题目的比例相同。在 100 步之内，知道哪些可见测试通过是错误的，并没有教会策略解出更多题目；它教会的是在解不出时更少地通过可见测试。一个对正确性完美的评判模型作为因子，给出的恰好就是这个奖励。

局限。 只有一种策略规模 (4B)、一种任务格式 (单个函数，最多三次尝试，以第一个失败的测试作为反馈)、100 步、一个取自第二部分的学习率，以及每种奖励两个种子；在 MBPP+ 上两个种子的标准差高达 4.9 个百分点。区间对题目重采样，不对种子重采样。留出测试是 KodCode 和 EvalPlus 的测试，一个程序通不过它们，是按它们的规格判定为错，而这种规格可能比题目描述更严格。特判检查只能发现含有可见测试中数值的程序；不含此类数值的特判（比如按输入长度分支）检测不到。训练期间其他实验共用 GPU。

可复现性

代码、提示、数据准备 (题目筛选和可见/留出测试的划分)、原始日志 (每个 GRPO 步和最终评估的每个回合)，以及离线研究的程序和评判分数都在仓库中 (`experiments/part3a_code/`)。表格和正文中的每个数字都由 `dcjev/code/judge_study.py`、`dcjev/analysis/part3a.py` 和 `dcjev/analysis/paper_numbers.py` 从中生成。

参考文献

- [1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- [2] Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025.
- [3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared

- Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [4] Carson Denison, Monte MacDiarmid, Fazl Barez, David Duvenaud, Shauna Kravec, Samuel Marks, Nicholas Schiefer, Ryan Soklaski, Alex Tamkin, Jared Kaplan, et al. Sycophancy to subterfuge: Investigating reward-tampering in large language models. *arXiv preprint arXiv:2406.10162*, 2024.
- [5] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023.
- [6] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, 2022.
- [7] Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kod-Code: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv preprint arXiv:2503.02951*, 2025.
- [8] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.