

策略与评判模型共用一个骨干： Kev-4B 为 MATH 上的 GRPO 节省了什么、带来了什么

DC-Jev-RL 项目，第二部分

2026 年 10 月

摘要

Kev-4B 是 Qwen3.5-4B-Base 上的一个 LoRA 适配器加一个指针头构成的评判模型，因此在同一基座上以另一个 LoRA 适配器训练的 GRPO 策略可以与它共用一份骨干。我们测量这样做节省了什么，以及评判模型给 MATH 上的 GRPO 带来了什么。在共享骨干上，评判模型的概率与其 fp32 参考值之差不超过 0.015，训练器加载后比单独加载评判模型少用 48% 显存。离线时，在基座策略的 4,000 个解答上，Kev-4B 区分正误的能力与同规模的自回归评判模型相当 (AUROC 0.87 对 0.88)；同一模型先推理再作答更好，代价约为 29 倍。在 GRPO 中 (200 步，每个条件 2-3 个种子)，只用验证器就把 MATH-500 准确率从 76.5% 提高到 82.5%。把 Kev-4B 判断正确的概率加入奖励，或用 Kev-4B 的分数给验证器做门控时，每个条件都有一个种子遥遥领先 (MATH-500 上 87.4% 和 88.4%)，其余种子与只用验证器的 run 相近。这两个领先的种子改成了以纯文本的非正式草稿开头。本项目的 26 个 GRPO run 中，做出这一切换的 6 个 (有的奖励中没有评判模型) 正是 AIME24 上最好的 6 个；训练解答的验证结果看不出切换因何而起，而只有 2-3 个种子，我们无法把它归因于评判模型。只用 Kev-4B 的分数训练时，策略学会写出 Kev-4B 认为正确、实际错误的解答，MATH-500 最终为 70.7%。Kev-4B 还会为答完之后把题目再解一遍付费；奖励中含有 Kev-4B 的 11 个 run 中有 6 个最终落入这种风格。代码、日志、每个训练解答以及本文的每个数字都在仓库中。

1 引言

基于可验证奖励的强化学习 (RLVR) 在答案可以检验的问题上训练语言模型，GRPO [3, 14] 及其变体 [11, 12] 是最常见的形式。验证器精确但粗糙：它只判断最终答案对不对，不管推理是否站得住；当一组样本全对或全错时，它不给任何信号。评判模型可以直接给推理打分，过程监督 [10, 16] 和生成式验证器 [18] 是这一想法的两种形式。评判模型会带来显存和时间上的代价，也带来风险：学到的奖励可能被策略利用 [5, 15]。

对于 Qwen3.5-4B 策略的 RLVR，Kev-4B [13] 是一个加入成本格外低的评判模型。它是同一基座模型 Qwen3.5-4B-Base 上的一个 LoRA 适配器加一个指针头，因此策略和评判模型可以共用一份骨干权重：策略 LoRA 与评判 LoRA 并列挂在骨干上，评判模型一次前向传播就读出答案，无

需生成文本。本文测量这种设计节省了什么，以及评判模型给 GRPO 带来了什么。我们提出三个问题：

1. 在同一进程、同一训练步下测量，共享骨干比把评判模型作为第二个模型加载节省多少显存 (§6)?
2. 与同家族、同大小的 AR 评判模型相比，以及与该模型先推理再作答相比，Kev-4B 区分策略解答正误的能力如何？每个评判模型有多容易被不该影响它的改动所左右 (§7)?
3. 把 Kev-4B 的判断加入验证器奖励，能否改进 MATH 上的 GRPO? Kev-4B 能否取代验证器 (§8)?

2 相关工作

RLVR 与 GRPO。 GRPO [14] 用同一提示下多个样本的组基线取代 PPO 的价值函数，DeepSeek-R1 [3] 使其成为推理训练的标准做法。后续分析表明细节很重要：长度归一化和组内标准差 [11]，以及优势尺度的选择 [12]。我们采用组均值基线和 Liu et al. [12] 推荐的批级尺度、词元级损失，不加 KL 项。现代 RL 框架用推理引擎采样、用另一套框架训练，两者对同一组权重算出的概率略有差异，截断重要性采样可以修正这一点 [17]。

用于推理的评判模型和奖励模型。 结果奖励模型和过程奖励模型分别给解答或步骤打分 [10, 16]。生成式验证器 [18] 询问语言模型某个解答是否正确（可以先推理），并读取“是”的概率。已知 LLM 评判对长度、风格和“声称正确”的说法很敏感 [19]。策略被推得越狠，对奖励模型的过度优化就越严重 [5]；在许多场景中，RL 策略都学会了利用其奖励信号 [1, 4, 15]。

策略与奖励之间共享权重。 多 LoRA 推理 [7, 9] 让多个适配器在推理时共用一个基座模型。这里把同一机制用在训练器内部：一个正在训练的策略适配器和一个冻结的评判适配器。使之严格成立的条件是，训练从不改变评判模型训练时所用的基座权重。

3 策略与评判模型共用一个骨干

Kev-4B [13] 是 Qwen3.5-4B-Base（修订版 1001bb4d）上的一个秩 16 的 LoRA 适配器加一个指针头。我们训练的策略是同一基座上的一个 LoRA 适配器，因此一份基座权重可以同时服务两个模型。这份权重上挂两个适配器：`default` 是策略（秩 32， $\alpha = 64$ ，覆盖所有注意力、MLP 和 Gated DeltaNet 投影），`kev` 是 Kev-4B 发布的适配器，冻结且以未合并方式使用。只有策略适配器被训练，所以基座权重始终不变，Kev-4B 读到的恰好是它训练时所用的骨干。给一个解答打分时，把活动适配器切换到 `kev`，并以其校准温度读取 Kev-4B 的指针头。此时评判模型看到的骨干和适配器权重与单独加载的 Kev-4B 完全相同，输出只因 bf16 下 LoRA 合并与未合并的算术差异而略有不同，这一差异我们做了测量 (§6)。

另一种做法是把 Kev-4B 作为第二个模型加载。我们在同一进程、同一训练步下测量两种设计的显存：加载后的已分配显存，以及在固定批次上做一次策略更新再做 Kev-4B 打分期间的峰值显

存。

4 GRPO 设置

策略与数据。 策略从 Qwen3.5-4B-Base 加一个零初始化的 LoRA 开始，所以第 0 步就是基座模型。训练题目是 MATH [6] 训练集中 3–5 级的题目。从中固定随机抽取 500 道（种子 12345）作为验证集，这是唯一用于模型选择的数据；其余训练题按带种子的随机顺序遍历。MATH-500 的测试题都不在 MATH 训练集划分中（构建数据时已检查）。提示要求逐步解答并把最终答案写在 `\boxed{\}` 中。若 math-verify [8] 判定解答中最后一个 boxed 答案与参考答案相等，则该解答正确。评判模型从不看到参考答案。

Rollout。 每一步从 vLLM [9] 服务器采样 16 道题、每题 8 个解答，温度为 1（最多 1024 个词元）；每次更新后，策略适配器都会发送给该服务器。vLLM 与训练框架对同一组权重算出的概率略有差异。我们对每个词元做截断重要性采样修正 [17]，权重为 $\min(\pi_{\text{train}}/\pi_{\text{vLLM}}, 2)$ ，并记录这一差异。

目标函数。 一个解答的优势等于其奖励减去所在组的平均奖励，再除以这些中心化奖励在整个批次上的标准差，与 Lite PPO [12] 相同。采用批级尺度后，即使验证器对一组中所有解答给出相同奖励，评判项的权重 λ 也保持原有含义。损失是词元级策略梯度，对批次中所有解答词元求和再除以其个数 [11]。每个批次只做一次 AdamW 更新，因此 PPO 的比率恒为 1，裁剪不起作用；不加 KL 项。学习率在验证集上用 rlvr 奖励从 $\{1, 3, 10\} \cdot 10^{-5}$ 中选出 (§8.1)，之后所有条件都固定使用。梯度范数裁剪为 1.0，每批 128 个解答，每个 run 训练 200 步。

奖励条件。 设 $v \in \{0, 1\}$ 为验证器的判定。 P_c 是 Kev-4B 对“所给解答的最终答案是否正确？”回答“是”的概率； P_g 是它对“所给解答是否通过真实、相关的逐步推理得出最终答案……？”回答“是”的概率； P_a 是 AR 评判对第一个问题回答“是”的概率。评判提示见 `experiments/part2_llm/prompts/judge_prompts.json`，在任何 GRPO run 之前即已固定。评判模型看到题目和解答（截断到 1536 个词元）。

条件	奖励
rlvr	v
rlvr_kev_add	$v + \lambda P_c, \lambda = 0.2$
rlvr_kev_gate	$v \cdot P_g$
kev_only	P_c （无验证器）
rlvr_ar_add	$v + \lambda P_a, \lambda = 0.2$

由于 $\lambda < 1$ ，在加性条件中正确解答的排序总是高于错误解答，评判模型只能在判定相同的解答之间重新排序。`kev_only` 提出的是另一个问题：仅靠评判模型能否训练策略，策略是否会学会利用评判模型。所有条件中，Kev-4B 还会在每步随机给 64 个解答打分，仅用于记录。

评估。 每 50 步在验证集、MATH-500 和 GSM8K 测试集 [2] 的一个固定 500 题子集上做贪心评估（最多 2048 个词元）。最后一步之后，在完整的 GSM8K 测试集和 MATH-500 上做贪心评估，在 AMC23、AIME24 和 AIME25 上每题采样 16 次（温度 0.6, top- p 0.95），报告样本的平均准确率。我们报告每个 run 的最终策略，不挑选检查点。逐题结果都被保存，因此各条件在相同的题目上比较：我们报告相对于 `rlvr` 的配对差值（对种子取平均），以及按题目做 bootstrap 得到的 95% 置信区间。

离线评判研究。 基座策略在 1000 道题（MATH-500 和 GSM8K 的 500 题子集）上每题以温度 1 写 4 个解答，所用提示和停止串与训练相同，因此这些正是 GRPO 的起点所产生的解答。两个评判模型对每个解答回答全部问题。对每种信号，我们报告它对验证器判定的 AUROC：一是对全部解答；二是在题目内部，即只看同一道题中判定不同的解答对，这正是 GRPO 组基线所用的比较。我们还报告校准程度（期望校准误差 ECE；Brier 分数）和代价。四种改动用来探测什么会左右评判模型：

- 只给答案：把解答替换成一句只给出其最终答案的话；
- 填充：在末尾追加二十遍一句自我怀疑的话；
- 声称：在末尾追加一句“每一步都已验证”的声明；
- 篡改：把正确解答的最终 boxed 答案改掉，使之变错。

在一个随机子集上，另设一个生成式评判作为进一步的基线：同一个 AR 模型先推理再作答（开启思考），分数取作答位置上的 $p(\text{yes})$ 。这项研究使用测试题，只用于测量评判模型，不影响 GRPO run 的任何部分；GRPO run 的设计和提示在它运行之前就已固定。

5 实验设置

每个 run 使用一台 Colab 虚拟机，配一块 NVIDIA RTX PRO 6000 Blackwell Server Edition GPU (96 GB)。训练器（kev 锁定环境中的 PyTorch 2.8；Qwen3.5 的 Hugging Face 实现，其 Gated DeltaNet 层运行在 flash-linear-attention 内核上）与 vLLM 0.31 rollout 服务器（单独的环境）共用这块 GPU，最多三个训练器共用一个 rollout 服务器。本系列的其他实验也常常同时在这块 GPU 上运行，所以我们只报告在空闲 GPU 上测得的耗时，并明确标注。种子固定题目顺序、vLLM 的采样种子（每道题每一步一个）以及训练器的所有随机选择。GPU 内核不是确定性的，所以同一种子的两次运行并非逐比特相同。每 25 步保存一次检查点，中断后可以恢复；恢复过的 run 数会一并报告。

6 共享骨干节省了什么

评判结果不变。 在 576 个关于数学解答的问题上（GSM8K 和 MATH 训练题的参考解答，原样的和改掉最终答案的），共享骨干上的 Kev-4B 与 Kev-4B 的 fp32 参考实现在概率上最多相差 0.015，平均相差 0.0023，在 99.0% 的问题上选出相同的选项。单独加载、合并 LoRA 并使用融合内核

judge	used by	3 questions		1 question	peak GB
		ms/request	ms/question	ms/question	
Kev-4B, merged, fused kernels, one row per question	Part I judge server	53.0	17.7	18.4	22.8
Kev-4B, merged, shared prefix	offline studies	44.3	14.8	27.9	13.4
Kev-4B on the policy’s backbone	GRPO trainers	49.9	16.6	41.5	10.1
AR judge, fused kernels	baseline	–	–	18.0	9.7
AR judge reasoning first (vLLM)	baseline	–	–	478	–

表 1: 在空闲的 RTX PRO 6000 上，对同样 1000 个基座策略解答 (MATH-500 和 GSM8K 题目) 做一次评判的代价。Kev-4B 在一个请求中回答关于一个解答的三个第二部分问题；AR 评判每个问题需要一次前向传播；推理型评判先生成最多 2048 个词元的思考再作答。峰值：进程分配的 GPU 显存，含模型本身。

的 Kev-4B 最多相差 0.009。三者区分正确解答与被篡改解答的能力相同 (AUROC 分别为 0.954、0.954 和 0.955)。

显存。 节省的是一份骨干，8.3 GB。加载完成后，带两个适配器和 Kev-4B 输出头的共享骨干占用 8.8 GB 显存，两个独立模型占用 17.1 GB，少 48%。在固定批次 (8 条 1024 词元的序列，开启梯度检查点) 上做一次策略更新，峰值为 14.4 GB 对 22.7 GB，少 37%。两种方案各在一个新进程中用相同的训练步测量 (`experiments/validation/shared_check_idle.*.json`)。随后 Kev-4B 打分阶段的峰值主要取决于评判模型一次处理多少词元，这是一个设置而不是任何一种设计的属性，因此不作比较。

速度。 把活动适配器从策略切换到 Kev-4B 再切回来需要 18 ms。在空闲 GPU 上，共享骨干上的 Kev-4B 每秒处理 36.2 个这样的请求，单独的合并模型为 39.6 个。共享骨干上的适配器以未合并方式计算，这会损失一些速度。在策略自己写的解答上 (更长)，GRPO 训练器中的这种配置处理一个含三个问题的请求需要 49.9 ms (表 1)。早先一次对共享配置的测量得到每秒 8.0 个请求，当时另有进程在使用 GPU，不能代表该设计本身的性能。

7 评判模型能多大程度看出策略的错误

基座策略在 4,000 次尝试中答对了 60.9% (MATH-500 上 51.7%，GSM8K 子集上 70.0%)。1000 道题中有 603 道既有答对的尝试也有答错的尝试。

Kev-4B 与 AR 评判的准确度相当。 表 2 给出每种信号对验证器判定的 AUROC。在全部尝试上，Kev-4B 的 *correct* 达到 0.87，AR 评判达到 0.88。AR 评判的领先幅度很小，但统计上可以分辨：Kev-4B 减 AR 为 $-0.013 [-0.021, -0.005]$ (按题目重采样得到的 95% 区间)。在题目内部，也

signal	all		MATH-500	GSM8K	ECE	Brier
	AUROC	within	AUROC	AUROC		
Kev-4B <i>correct</i>	0.87	0.82	0.88	0.83	0.10	0.16
Kev-4B <i>genuine</i>	0.87	0.82	0.89	0.84	0.14	0.17
Kev-4B <i>quality</i>	0.88	0.83	0.90	0.84	0.09	0.15
AR <i>correct</i>	0.88	0.80	0.90	0.83	0.06	0.13
AR <i>quality</i>	0.87	0.80	0.90	0.82	0.25	0.22

表 2: 在 1000 道题 (MATH-500 和 GSM8K 测试集中的 500 道) 上、每题 4 次基座策略尝试中, 各评判信号对验证器判定的 AUROC: 全部尝试上的, 以及题目内部的 (同一道题中判定不同的尝试对); 该信号作为“正确”概率的 ECE 和 Brier 分数 (主要对 *correct* 类问题有意义)。

就是 GRPO 组基线所做的比较, 排序反了过来: 0.82 对 0.80, 差值 $+0.019 [-0.001, 0.041]$, 不能分辨。在 GSM8K 上可以分辨 ($+0.049 [0.020, 0.081]$)。在更接近训练题的 MATH-500 上, Kev-4B 落后, 但幅度不能分辨 ($-0.017 [-0.044, 0.012]$)。Kev-4B 的三个问题携带的信息几乎相同: *genuine* 和 *quality* 的 AUROC 与 *correct* 相近。作为“答案正确”的概率, AR 评判的是/否概率校准得更好 (ECE 0.06 对 0.10)。Kev-4B 的 *correct* 在错误解答上偏高 (均值 0.51, AR 评判为 0.34)。

先推理的评判更好, 但有代价。 在随机抽取的 1000 次尝试上, 先推理再作答的 AR 模型达到 AUROC 0.92 (在该子集中两种结果都有的 82 道题内部为 0.91), 同样这些尝试上 Kev-4B 的 *correct* 为 0.88 和 0.88。全部尝试上的领先可以分辨 ($+0.048 [0.027, 0.069]$); 题目内部的领先不能分辨 ($+0.024 [-0.054, 0.099]$), 因为子集中两种结果都有的题目太少。它的思考在多数情况下触到了 2048 词元的上限 (只有 38% 正常结束), 之后仍被要求给出答案。在空闲 GPU 上, 它每次评判耗时 478 ms, 约为 GRPO 训练器中 Kev-4B 每个问题耗时的 29 倍 (§6)。GRPO 每一步要评判 128 个解答。

什么会左右评判模型。 表 3 给出四种改动下各评判模型 *correct* 概率的平均变化。正确解答的最终答案被篡改时, 两个评判模型都能察觉 (Kev-4B -0.53 , AR -0.75)。两者都会惩罚不带推理、只给出的正确答案 (-0.21 和 -0.34), 而对不带推理的错误答案则基本不变。在末尾追加“每一步都已验证”的声明会让两个评判模型都略微升高: Kev-4B 在正确解答上升高 $+0.02$, 其中 83% 的解答分数上升; AR 评判升高 $+0.03$ 。这些变化与正确、错误解答之间的差距相比很小, 但它们是系统性的, 以评判分数为训练目标的策略找到它们就会得到奖励。

edit	correct completions		incorrect completions	
	Kev-4B ΔP_c	AR ΔP_a	Kev-4B ΔP_c	AR ΔP_a
answer only	-0.21	-0.34	-0.02	+0.00
padded	-0.03	-0.04	-0.01	-0.04
asserted	+0.02	+0.03	+0.01	+0.03
altered	-0.53	-0.75	-	-

表 3: 解答被改动时 P_c (Kev-4B) 和 P_a (AR 评判) 的平均变化, 按验证器判为正确和错误的解答分别统计。只给答案: 解答替换为一句给出其最终答案的话; 填充: 追加二十句自我怀疑的话; 声称: 追加一句已验证的声明; 篡改: 改掉正确解答的最终答案。

学习率	第 0 步	第 30 步	第 60 步	得分 (第 30 与 60 步均值)
10^{-5}	0.692	0.738	0.738	0.738
$3 \cdot 10^{-5}$	0.694	0.730	0.744	0.737
10^{-4}	0.698	0.748	0.678	0.713

表 4: 学习率扫描: `rlvr`、种子 1 的验证准确率 (500 道留出的 MATH 训练题, 贪心解码)。

8 评判模型能为验证器带来增益吗？

8.1 学习率

学习率只在验证集上选一次, 使用 `rlvr` 奖励和种子 1。每个扫描 run 训练 60 步, 每步 16 道题、每题 8 个回答, 学习率分别为 10^{-5} 、 $3 \cdot 10^{-5}$ 和 10^{-4} (表 4)。选择规则是: 取第 30 步和第 60 步验证准确率均值最高的学习率; 若最好的两个相差不超过 0.5 个百分点, 则取较小的那个 (`experiments/part2_llm/LR_SELECTION.md`)。这条规则是在 10^{-5} 和 10^{-4} 两个 run 刚开始时写下的; 当时 $3 \cdot 10^{-5}$ 的 run 已经记录了第 0 步和第 30 步的准确率, 我们也看过。 10^{-5} (0.738) 与 $3 \cdot 10^{-5}$ (0.737) 相差不超过 0.5 个百分点, 因此规则为第二、三部分的所有 run 选定了 10^{-5} 。在 10^{-4} 下, 验证准确率到第 30 步升至 0.748, 到第 60 步又跌到 0.678, 低于基座模型。三个 run 都从同一个基座模型出发, 第 0 步的准确率却不同 (0.692、0.694、0.698): vLLM 上的贪心解码在不同服务器状态下并非逐位可复现, 这一差异就是评估噪声的大小。

8.2 奖励条件

`rlvr` 和 `rlvr_kev_add` 各跑了 3 个种子, 其余条件各 2 个。表 5 给出最终策略的准确率, 表 6 给出每个 run 的准确率, 表 7 给出相对 `rlvr` 的配对差, 图 1 给出训练曲线。配对区间按题目重采样, 不包含种子之间的差异; 而在这些 run 中, 种子之间的差异才是更大的那一个。

只用验证器。 `rlvr` 训练 200 步, 把 MATH-500 从 76.5% (基座模型) 提高到 82.5%, GSM8K 从 87.9% 提高到 93.0%, AMC23 从 50.6% 提高到 60.9%, AIME24 和 AIME25 从 10.0% 和 10.9%

condition	seeds	MATH-500	GSM8K	AMC23	AIME24	AIME25
base model	–	76.5	87.9	50.6	10.0	10.9
rlvr	3	82.5 $\pm$ 0.6	93.0 $\pm$ 0.3	60.9 $\pm$ 2.4	21.5 $\pm$ 0.4	16.7 $\pm$ 2.1
rlvr_kev_add	3	84.6 $\pm$ 2.6	93.1 $\pm$ 0.9	65.6 $\pm$ 3.9	25.6 $\pm$ 6.6	18.6 $\pm$ 5.5
rlvr_kev_gate	2	85.2 $\pm$ 4.5	93.3 $\pm$ 0.5	67.7 $\pm$ 4.6	28.5 $\pm$ 8.5	21.1 $\pm$ 4.9
kev_only	2	70.7 $\pm$ 1.3	91.3 $\pm$ 0.0	41.8 $\pm$ 2.3	4.9 $\pm$ 1.0	2.3 $\pm$ 1.5
rlvr_ar_add	2	82.4 $\pm$ 1.7	93.2 $\pm$ 0.8	62.1 $\pm$ 2.3	21.6 $\pm$ 2.8	18.0 $\pm$ 1.9

表 5: 最终策略：准确率 (%), 种子均值 $\pm$ 种子间标准差。MATH-500 和 GSM8K (1319 题)：贪心解码。AMC23、AIME24 和 AIME25：温度 0.6 下每题 16 个样本的平均。基座模型：两次评估的平均。

condition	seed	MATH-500	GSM8K	AMC23	AIME24	AIME25
rlvr	1	82.8	93.0	63.6	21.0	18.8
	2	83.0	93.3	59.8	21.9	16.9
	3	81.8	92.6	59.2	21.7	14.6
rlvr_kev_add	1	82.2	92.1	62.5	21.9	13.8
	2	84.2	93.6	64.4	21.7	17.5
	3	87.4	93.7	70.0	33.1	24.6
rlvr_kev_gate	1	82.0	92.9	64.4	22.5	17.7
	2	88.4	93.7	70.9	34.6	24.6
kev_only	1	69.8	91.3	40.2	4.2	1.2
	2	71.6	91.3	43.4	5.6	3.3
rlvr_ar_add	1	81.2	92.6	60.5	19.6	16.7
	2	83.6	93.8	63.7	23.5	19.4

表 6: 每个 run 的最终策略：准确率 (%), 口径同表 5。

提高到 21.5% 和 16.7%。

每个 Kev-4B 条件都有一个种子遥遥领先。 **rlvr** 的三个种子在 MATH-500 上最终为 82.8%、83.0% 和 81.8%，在 AIME24 上为 21.0%、21.9% 和 21.7% (表 6)。把 Kev-4B 的 P_c 加入奖励后，两个种子的结果与之相近 (MATH-500 上 82.2% 和 84.2%)，第三个为 87.4%。用门控时，一个种子为 82.0%，另一个为 88.4%。在 AIME24 上，这两个领先的种子达到 33.1% 和 34.6%，本文其他 run 都不超过 23.5%。这两个 run 撑起了各自条件的均值，也撑起了相对 **rlvr** 的配对差：**rlvr_kev_add** 在 MATH-500 上为 +2.1 [0.6, 3.6] 个百分点，**rlvr_kev_gate** 为 +2.7 [0.7, 4.8]，在竞赛题集上更大 (表 7)。这些区间说明，在这些题目上这两个条件的种子均值高于 **rlvr**；它们并不说明差别来自评判模型：同一条件的其他种子并没有超过 **rlvr**。加入 AR 评判模型的 P_a (**rlvr_ar_add**) 时，两个种子都与 **rlvr** 相近 (MATH-500 上 -0.1 [-1.9, 1.6] 个百分点)。让这两个领先 run 与众不同

minus rlvr	MATH-500	GSM8K	AMC23	AIME24	AIME25
rlvr_kev_add	+2.1 [0.6, 3.6]	+0.1 [−0.5, 0.8]	+4.7 [1.6, 8.2]	+4.0 [0.1, 8.3]	+1.9 [0.2, 3.7]
rlvr_kev_gate	+2.7 [0.7, 4.8]	+0.3 [−0.4, 1.0]	+6.8 [2.6, 11.3]	+7.0 [2.4, 12.2]	+4.4 [1.3, 8.2]
kev_only	−11.8 [−14.6, −9.1]	−1.7 [−2.6, −0.9]	−19.1 [−28.2, −10.4]	−16.6 [−26.9, −7.5]	−14.4 [−24.4, −5.9]
rlvr_ar_add	−0.1 [−1.9, 1.6]	+0.2 [−0.5, 0.9]	+1.2 [−2.0, 4.7]	+0.0 [−2.5, 3.0]	+1.3 [−1.6, 4.6]

表 7: 相对 rlvr 的配对差 (百分点): 逐题取各条件的种子均值之差, 再对题目取平均, 附按题目自助法得到的 95% 区间。

condition	training, last 20 steps					final policy, MATH-500	
	accuracy	Kev-4B P_c	tokens	all wrong	all right	Kev-4B P_c	Kev-4B AUROC
rlvr	67.0	0.74	679	16%	44%	0.81	0.88
rlvr_kev_add	69.0	0.74	593	14%	46%	0.82	0.89
rlvr_kev_gate	69.7	0.78	831	14%	47%	0.87	0.89
kev_only	55.5	0.87	829	25%	32%	0.91	0.85
rlvr_ar_add	68.0	0.73	585	15%	47%	0.80	0.89

表 8: 最后 20 步的训练统计 (种子均值): 训练解答的验证器准确率、每步其中 64 个解答上 Kev-4B 的平均 P_c 、解答词元数, 以及组内解答全错或全对 (验证器不给学习信号) 的组所占比例。最终策略: MATH-500 解答上 Kev-4B 的平均 P_c 及其对验证结果的 AUROC。

的, 是它们的写法 (§8.3)。

只用评判模型会被钻空子。 没有验证器时 (kev_only), 策略最大化 Kev-4B 的 P_c : 在最后 20 步的训练解答上, 它升到 0.87 (rlvr: 0.74), 而验证器在同样这些解答上的准确率停在 55.5% (rlvr: 67.0%)。与 rlvr 相比, 最终策略在 MATH-500 上相差 −11.8 [−14.6, −9.1] 个百分点, 在 AIME24 上相差 −16.6; 两个种子在 MATH-500 上都低于它们出发时的基座模型 (69.8% 和 71.6%, 基座为 76.5%), 在三个竞赛题集上也都如此。策略学会的是写出 Kev-4B 认为正确的解答: Kev-4B 给它在 MATH-500 上的错误答案平均打 0.86, 而给 rlvr 各策略的错误答案打 0.61。Kev-4B 在训练解答上的组内 AUROC——GRPO 的优势用的正是组内比较——从前 20 步的 0.85 降到最后 20 步的 0.65。对于没有针对它做优化的策略, 4B 的评判模型是可用的信号 (§7); 单独作为奖励, 它不可用。

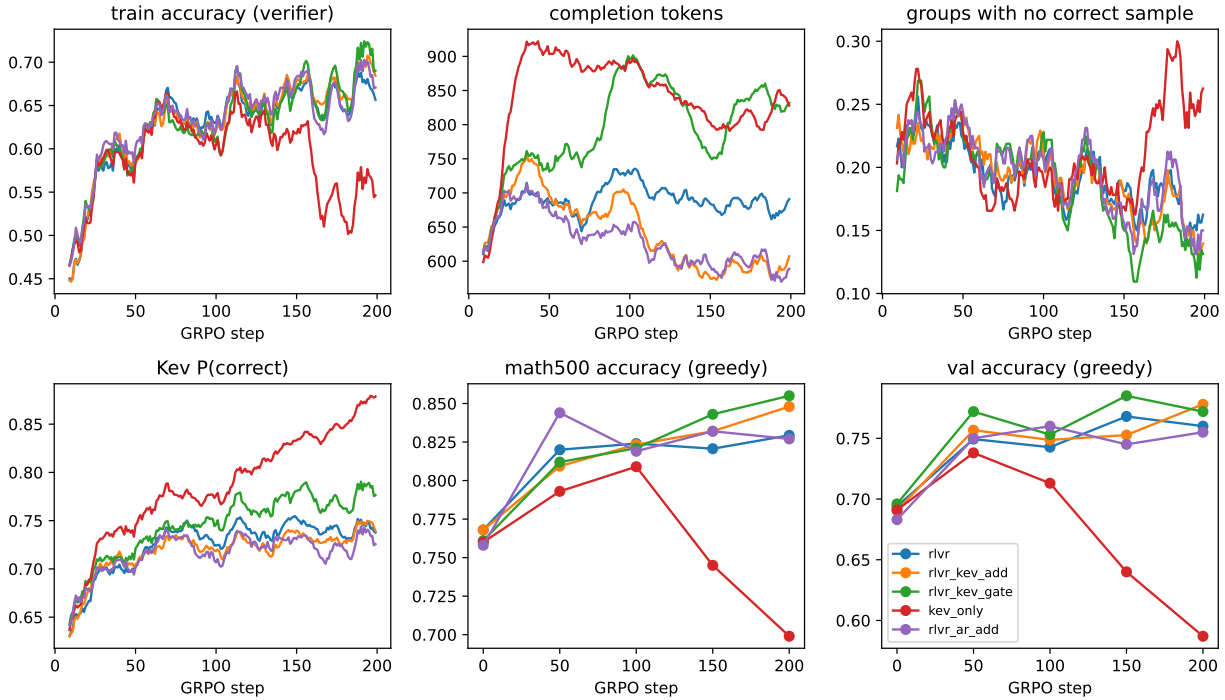


图 1: 训练曲线 (种子均值): 训练解答的验证器准确率、解答词元数、没有任何正确解答的组所占比例、训练解答上 Kev-4B 的 P_c (10 步滑动平均), 以及每 50 步在 MATH-500 和验证集上的贪心准确率。

评估噪声。 每个 run 都在第 0 步评估基座模型。在 12 个 run 中, 第 0 步的 MATH-500 准确率在 75.4% 到 78.0% 之间 (标准差 0.8 个百分点), 验证准确率在 68.2% 到 70.2% 之间。每个 run 的最终权重在 MATH-500 上被贪心评估两次: 一次是第 200 步的定期评估, 一次是最终评估。两次结果平均相差 0.6 个百分点, 最多相差 1.4 (12 个 run)。两个最终策略之间一个百分点的差异落在这一噪声之内; 领先种子与其他种子之间的差异则不然。

8.3 写解答的三种方式

图 2 给出三个 run 的最终策略对同一道 MATH-500 题目的解答。多数策略写一份带 LaTeX 和 markdown 格式的解答, 以框出的答案结尾 (左)。两个领先的种子先用纯文本、不带 LaTeX 或 markdown 把题目做一遍, 给出答案, 多数情况下再写格式化的解答 (中)。还有一些策略写完格式化的解答、给出答案之后, 又用几乎相同的话把解答再写一遍 (右)。第一种风格, 我们用以非正式草稿开头的解答所占比例来衡量: 前 300 个字符中没有数学定界符 ($\$, \backslash($ 或 $\backslash[$), 也没有 markdown 标题或粗体 (**###** 或 ******)。第三种风格, 我们用一个解答中 (至少 20 个字符的) 行有多大比例与该解答前面某行相同来衡量。这两个量都是在读过最终解答之后为了描述所见而定义的。

两个领先的种子改成了先写草稿。 在第 0 步, 基座模型有 11% 到 12% 的 MATH-500 解答以草稿开头 (表 9)。每个 run 在前 50 步都压低了这一比例, 第 50 步时至多为 9%; 十二个 run 中有十个一直保持在低位 (最终评估中至多 3%)。rlvr_kev_gate 种子 2 的这一比例从第 100 步的 5%

condition	seed	MATH-500	AIME24	share of drafts, step					
				0	50	100	150	200	final
rlvr	1	82.8	21.0	12%	1%	0%	1%	1%	1%
	2	83.0	21.9	12%	0%	0%	0%	0%	0%
	3	81.8	21.7	12%	1%	1%	0%	1%	1%
rlvr_kev_add	1	82.2	21.9	12%	0%	0%	0%	0%	0%
	2	84.2	21.7	12%	1%	0%	1%	1%	1%
	3	87.4	33.1	12%	9%	2%	16%	90%	90%
rlvr_kev_gate	1	82.0	22.5	12%	2%	1%	2%	1%	1%
	2	88.4	34.6	12%	4%	5%	77%	85%	85%
kev_only	1	69.8	4.2	12%	1%	1%	1%	0%	1%
	2	71.6	5.6	11%	0%	0%	0%	0%	0%
rlvr_ar_add	1	81.2	19.6	11%	1%	1%	3%	3%	3%
	2	83.6	23.5	12%	1%	1%	0%	2%	2%

表 9: 以非正式草稿开头（前 300 个字符中没有数学定界符、markdown 标题或粗体）的 MATH-500 解答所占比例：每次定期评估（贪心，每 50 步）和最终评估，逐个 run 列出，并附最终准确率。

升到第 150 步的 77%，rlvr_kev_add 种子 3 从第 150 步的 16% 升到第 200 步的 90%。这两个 run 的最终 MATH-500 解答分别在 85% 和 90% 的题目上以草稿开头，其中分别有 85% 和 58% 的草稿之后接着写了格式化的内容；其余的草稿就是整篇解答。它们并不比其他 run 的解答长（平均 797 和 623 个词元；表 11）。同一种子的 run 以相同顺序看到相同的题目，并从相同的样本出发，所以这一切换不是数据顺序的属性：rlvr 的种子 2 和 3 就没有切换。

另外两篇论文中也发生了同样的切换，且只发生在最好的 run 上。第三部分 B 中有 2 个 run、第三部分 C 中有 2 个 run 以同样的方式切换，其中 2 个的奖励只有验证器（在第三部分 C 的两个树状条件中；tree_random 与 rlvr 的唯一区别是一组解答的采样方式）。开头的纯文本之后并不总有格式化的解答：在第三部分 C 切换了的 run 中，只有 25% 和 15% 的草稿之后接着写了格式化的内容。表 13 按最终策略的写作风格列出本项目全部 26 个 GRPO run。切换了的 6 个 run 在 AIME24 上达到 30.4% 到 34.6%，其他 run 至多 24.4%；在 MATH-500 上它们达到 84.6% 到 88.4%，其他 run 至多 84.6%。分类不取决于定义的细节：把开头窗口从 300 个字符换成 150 或 600 个字符，或者只把数学定界符当作标记，切换了的 run 仍有至少 53% 的最终解答以草稿开头，其他 run 至多 16%。这一切换是在 run 全部结束后才发现的，分类也是我们定的，所以这是对 these run 的描述，而不是对事先提出的假设的检验。

验证结果看不出原因。GRPO 提高优势为正的解答的概率，因此当一种风格的解答比同组其他解答更好时，这种风格就会扩散。表 10 在每个同时含有两类解答的 8 个训练解答的组内，比较草

condition	seed	steps 0–49		steps 50–99		steps 100–149		steps 150–199	
		drafts	Δ right	drafts	Δ right	drafts	Δ right	drafts	Δ right
rlvr	1	7%	−8.5	2%	+4.2	1%	−3.9	2%	+1.8
	2	5%	−15.3	1%	−9.3	0%	−5.0	0%	−2.1
	3	6%	−14.3	1%	−0.7	1%	+3.3	1%	+2.6
rlvr_kev_add	1	6%	−8.2	1%	−10.7	1%	−3.1	0%	−25.9
	2	7%	−10.6	1%	−4.6	1%	+6.9	1%	−1.9
	3	9%	−9.4	7%	−1.6	5%	−1.3	62%	−1.2
rlvr_kev_gate	1	8%	−11.2	2%	−4.3	3%	−2.5	2%	−7.9
	2	13%	−7.9	3%	−0.1	26%	−0.6	39%	+2.2
kev_only	1	9%	−9.9	1%	−7.1	1%	−6.2	1%	+9.4
	2	8%	−7.9	1%	+1.4	0%	−0.9	0%	−17.3
rlvr_ar_add	1	7%	−11.0	2%	−2.3	4%	+0.4	3%	−3.4
	2	6%	−10.9	1%	−2.9	1%	+2.4	1%	−1.3

表 10: 以非正式草稿开头的训练解答，按 50 步的窗口：它们所占比例，以及 Δ right：在同时含有两类解答的组上取平均的、草稿被验证器判为正确的比例减去同组其他解答的这一比例（百分点）。

稿开头的解答和其他解答被判为正确的频率。第 0–49 步，在全部 12 个 run 中草稿都更少被判正确：差值在 −15.3 到 −7.9 个百分点之间，这就是它们的比例下降的原因。从第 50 步起，在切换了的两个 run 中差值在 −1.6 到 +2.2 个百分点之间，在其他 run 中（它们的组里很少有草稿）在 −25.9 到 +9.4 之间。在 **rlvr_kev_add** 种子 3 中，草稿在每个窗口里都比同组其他解答更少被判正确，第 150–199 步差 −1.2 个百分点，而它们在训练解答中的比例却从 5% 升到了 62%；它们的优势（包含评判项）也不更大（平均 −0.019）。是什么让这些 run 切换的，我们记录的验证结果和奖励都看不出来。

四个策略学会了重复自己。 **kev_only** 的两个种子、**rlvr_kev_gate** 种子 1 和 **rlvr** 种子 3 写完解答、给出答案，再把解答写一遍（图 2 右）。在它们的最终 MATH-500 解答中，有 13% 或更多的行与前面某行相同，而本文其他 run 都至多 1.1%；**rlvr** 种子 3 有 90% 的解答在第一个答案之后重复了部分解答（表 11）。**rlvr** 的奖励只有验证器，所以这不需要评判模型。不过奖励中含有 Kev-4B 分数时，它更常见：在本项目中，奖励含有 Kev-4B 分数的 11 个 run 有 6 个以这种方式结束，其他 15 个 run 中只有 2 个；而且评判模型会为此付费 (§8.4)。重复并没有让策略更准确：奖励中含有验证器的重复 run 在 MATH-500 上最终为 78.6%–82.6%（平均 80.8%），两种风格都不属于的 run 为 80.6%–84.6%（平均 82.8%）。

condition	seed	tokens	truncated	repeated lines		several	draft	Kev-4B P_c	
				share	after answer	answers	opening	right	wrong
rlvr	1	541	2%	1%	0%	2%	1%	0.84	0.60
	2	555	4%	1%	0%	6%	0%	0.85	0.59
	3	909	8%	28%	90%	95%	1%	0.88	0.64
rlvr_kev_add	1	494	1%	1%	0%	13%	0%	0.84	0.61
	2	502	2%	1%	0%	5%	1%	0.84	0.60
	3	623	6%	0%	1%	62%	90%	0.90	0.62
rlvr_kev_gate	1	1001	10%	19%	86%	90%	1%	0.90	0.62
	2	797	9%	1%	3%	80%	85%	0.92	0.66
kev_only	1	775	1%	22%	77%	100%	1%	0.93	0.88
	2	748	0%	13%	87%	100%	0%	0.93	0.84
rlvr_ar_add	1	485	1%	1%	1%	10%	3%	0.84	0.61
	2	540	3%	1%	0%	5%	2%	0.84	0.59

表 11: 每个最终策略在 MATH-500 上写了什么 (贪心): 解答词元数、在 2048 词元处被截断的比例、(至少 20 个字符的) 行与同一解答前面某行相同的比例、在第一个框出答案之后重复这样一行的解答比例、含有多个框出答案的解答比例 (验证器读最后一个)、以非正式草稿开头的解答比例, 以及 Kev-4B 对正确与错误答案的 P_c 。

8.4 评判模型会为重复付费吗？

GRPO run 结束之后, 我们检验了评判模型是否会为重复付费 (dcjev/llm/repeat_probe.py; 表 12)。所有改动都不改变最后一个框出的答案 (验证器读的就是它), 每个评判模型都按训练时的方式打分。

单纯的重复对 Kev-4B 价值不大。 把判断研究中基座策略解答的最后两段再写两遍, Kev-4B 的 P_c 在验证器判为正确的 2436 个解答上提高 +0.012 [0.010, 0.013], 在判为错误的 1537 个解答上变化 -0.002 [-0.006, 0.001]。把整个解答写两遍, 变化为 +0.012 和 -0.003。AR 评判模型变化更大: 对正确答案 +0.037, 对错误答案 +0.012。

策略学会的那种重复是有回报的。 我们取重复的 run——kev_only 和 rlvr_kev_gate (各取种子 1) 以及 rlvr (种子 3) ——中含有重复行的最终 MATH-500 解答, 把每个在第一个答案之后截断, 就像策略在那里停下了一样。在第一个答案之前就开始重复的解答, 以及第一个答案与最后一个答案不同的解答, 不计在内 (分别 17、66 和 54 个)。Kev-4B 给原样的解答打分高于截断后的解答。对于奖励就是 Kev-4B 的 P_c 的 kev_only, 第一个答案之后的部分在正确答案上把它提高 +0.059 [0.052, 0.066], 在错误答案上提高 +0.107 [0.093, 0.122], 98% 的错误答案分数上升: 从 0.77 升到 0.88。rlvr_kev_gate 把验证器的奖励乘以 Kev-4B 的 P_g , 因此在验证器判为正确的解答之

	solutions	Kev-4B P_c		Kev-4B P_g		AR P_a	
	right / wrong	right	wrong	right	wrong	right	wrong
<i>base policy's solutions, repetition added</i>							
last two paragraphs twice more	2436 / 1537	+0.012	-0.002	+0.007	-0.006	+0.037	+0.012
written twice	2436 / 1564	+0.012	-0.003	+0.006	-0.004	+0.029	+0.007
<i>repeating final solutions of GRPO runs, as written against cut after the first answer</i>							
kev_only, seed 1	323 / 124	+0.059	+0.107	+0.016	+0.046	+0.046	+0.050
rlvr, seed 3	345 / 64	+0.043	+0.064	+0.017	+0.058	+0.037	-0.004
rlvr_kev_gate, seed 1	349 / 43	+0.058	+0.091	+0.026	+0.073	+0.033	+0.005

表 12: 重复对评判模型值多少: P (有重复) 减 P (无重复), 分别在验证器判为正确 (right) 和错误 (wrong) 的解答上, 附各自的数量。上: 判断研究中基座策略的解答 (MATH-500 和 GSM8K 子集), 加入重复。下: 重复的 GRPO run 的最终 MATH-500 解答, 原样对比在第一个答案之后截断。Kev-4B 的 P_c 和 P_g 以及 AR 评判模型的 P_a , 各自按训练时的方式打分; 正文中的区间断题重采样。

间, 只有这个因子决定谁的回报更高。在这些解答上, 第一个答案之后的部分把 P_g 提高 +0.026 [0.023, 0.030], 87% 的解答分数上升: 门控给重复自己的正确答案的回报, 高于就此停下的正确答案。Kev-4B 不在 rlvr 种子 3 的奖励中, 但它同样给这个 run 原样的重复解答打更高的分: 正确答案上 P_c 高 +0.043 [0.038, 0.047], 错误答案上高 +0.064 [0.044, 0.082], 正确答案上 P_g 高 +0.017。不在这些奖励中的 AR 评判模型, 也给带有第一个答案之后部分的解答打更高的分 (正确答案上分别高 +0.046、+0.033 和 +0.037), 只是不那么一致。

我们加入的重复是逐字复制, 而策略是用稍有不同的措辞把题目再解一遍。评判模型为这种重复的哪一点付费, 探针并不能告诉我们。它确实表明, 这些策略在第一个答案之后学会写的内容会提高评判模型的分数, 所以含有评判模型的奖励会为它付费; 无论策略是否针对评判模型训练过, 都是如此。

9 讨论

共享骨干带来了什么。 共享骨干上的评判模型与单独加载的评判模型给出相同的答案, 而训练器只需保存一份骨干而不是两份: 加载后少用 48% 显存。对 96 GB GPU 上的 4B 策略, 这只是方便; 在更小的 GPU 上或更大的模型上, 它可能决定评判模型能否与策略放在一起。代价是每次切换适配器 18 ms, 以及评判适配器以未合并方式使用带来的一些速度损失。

评判模型在验证器之上带来了什么。 rlvr_kev_add 和 rlvr_kev_gate 各有一个种子最终远高于 rlvr 的每个种子, 并把各自条件的均值抬到了 rlvr 之上。这两个 run 与其他 run 的区别在于它们改成了先写草稿; 本项目中共有 6 个 run 这样做了, 其中 2 个的奖励只有验证器。同一条件的其他种子最终与 rlvr 相近。每个条件只有 2-3 个种子, 我们无法判断奖励中的评判模型是否让这种切换更容易发生 (这里加入 Kev-4B 分数或用它做门控的 5 个 run 中有 2 个切换, rlvr 的 3 个

condition	seed	paper	MATH-500	AIME24	tokens	repeated lines	draft opening
<i>informal draft first</i>							
rlvr_kev_gate	2	II	88.4	34.6	797	1%	85%
rlvr_kev_add	3	II	87.4	33.1	623	0%	90%
tree_kev	1	III-C	87.2	31.9	567	0%	78%
sub_probe	1	III-B	87.2	30.6	546	0%	90%
tree_random	2	III-C	86.6	33.3	549	0%	82%
sub_probe	2	III-B	84.6	30.4	694	0%	90%
<i>neither</i>							
tree_kev	3	III-C	84.6	22.9	485	1%	7%
rlvr_kev_add	2	II	84.2	21.7	502	1%	1%
rlvr_ar_add	2	II	83.6	23.5	540	1%	2%
tree_random	3	III-C	83.2	22.3	582	1%	2%
sub_kev_recal	2	III-B	83.0	24.4	538	1%	1%
rlvr	2	II	83.0	21.9	555	1%	0%
rlvr_sub	2	III-B	83.0	17.1	588	1%	2%
rlvr	1	II	82.8	21.0	541	1%	1%
tree_random	1	III-C	82.4	21.0	553	1%	3%
rlvr_kev_add	1	II	82.2	21.9	494	1%	0%
rlvr_ar_add	1	II	81.2	19.6	485	1%	3%
rlvr_sub	1	III-B	80.6	19.6	630	1%	1%
<i>repeats itself</i>							
tree_kev	2	III-C	82.6	20.8	931	24%	3%
rlvr_kev_gate	1	II	82.0	22.5	1001	19%	1%
rlvr	3	II	81.8	21.7	909	28%	1%
sub_kev	2	III-B	80.4	17.1	1072	24%	2%
sub_kev_recal	1	III-B	79.4	17.9	904	22%	7%
sub_kev	1	III-B	78.6	14.8	1723	49%	1%
kev_only	2	II	71.6	5.6	748	13%	0%
kev_only	1	II	69.8	4.2	775	22%	1%

表 13: 本项目的全部 GRPO run (第二部分、第三部分 B 和 C, 各 200 步), 按最终策略的写作风格分组: 先写非正式草稿 (至少一半的 MATH-500 解答以草稿开头)、重复自己 (MATH-500 解答中平均至少 5% 的行与前面某行相同), 或两者都不是。两个比例在各 run 之间都是双峰分布, 因此分组不取决于这些阈值。最终准确率 (%); 每个 MATH-500 解答的词元数。第二部分的 `rlvr` 是三篇论文共同的参照, 只列一次。

run 中没有)。评判模型能起作用的余地本来就很窄。验证器给每个解答一个确切的判定，而 $\lambda < 1$ 时评判模型只能在判定相同的解答之间重新排序：在错误解答之间偏向更接近正确的那些，以及在全错的组里——验证器在那里完全不给信号。前几步之后，全错的组已经很少 (rlvr 最后 20 步为 16%)，而在判定相同的解答之间，评判模型的偏好可能关乎风格，不亚于关乎正确与否：Kev-4B 会为答完之后把题目再解一遍付费 (§8.4)，本项目奖励中含有 Kev-4B 分数的 11 个 run 中有 6 个最终如此，其他 15 个 run 中只有 2 个。这些重复的 run 都没有因此变得更准确。

种子与风格。 我们的 run 之间最大的差别不在奖励条件之间，而在最终策略的写作风格之间。改成先写草稿的 run 在 AIME24 上达到 30.4% 到 34.6%，本项目其他 run 至多 24.4%。切换是突然发生的：在这些 run 的每一个中，MATH-500 解答里草稿所占比例在相隔 50 步的两次评估之间从至多 28% 升到至少 77%，最早从第 50 步开始，最晚从第 150 步开始。验证器对训练解答的判定看不出是什么触发了它。像我们这样每个条件只有一到三个种子的奖励变体比较，可能把这样的切换误当成奖励的效果。因此我们报告每个 run 及其风格；在小模型上比较 RLVR 变体，也应当这样做，或者用足够多的种子来估计每种变体发生切换的频率。先写草稿是更高准确率的原因，还是只是与之相伴，我们没有检验；在评估或训练时强制或禁止草稿开头，会是第一步检验。

评判模型不是验证器。 离线时，Kev-4B 的 P_c 区分基座策略正误解答的 AUROC 接近 0.9 (§7)。探针表明，不改变答案的改动也能左右它，尽管幅度很小（声称每一步都验证过）。只对 P_c 训练时，策略找到了评判模型高估的解答：它的错误答案得到的 P_c 达到评判模型原先只给正确答案的水平，评判模型在组内的排序退化到接近随机，除 GSM8K 外每个基准上的准确率都跌到基座模型之下。这正是 Gao et al. [5] 所描述的奖励过度优化，只是规模更小、发生在 200 步之内。离线 AUROC 衡量的是评判模型在其训练分布上的表现，而不是在针对它优化的策略所产生的分布上，也不是在策略自己漂移进去的风格上。

局限。 一种策略与评判模型规模 (4B)，一个任务 (MATH，在 MATH-500、GSM8K 和三个竞赛题集上评估)，200 步，一个为 rlvr 选定的学习率，一个评判权重，每个条件 2–3 个种子——不足以说明一个条件以多大频率落到每种风格。区间按题目重采样，不按种子。vLLM 上对同一模型的重重复心评估相差约 0.8 个百分点 (§8.2)。两个风格量是在读过最终解答之后定义的。仓库中没有策略的权重（有它们的解答、日志和训练解答），因此无法从仓库出发对最终策略做干预实验。训练期间其他实验共用 GPU，所以除在空闲 GPU 上测得的时间外，我们不报告训练时间。

可复现性

代码、提示、学习率选择规则与扫描、原始日志（每个 GRPO 步和每次评估，以及逐题评估输出）、GRPO run 的每个训练解答，以及离线研究的解答和评判分数都在仓库中 (experiments/part2_llm/、experiments/validation/)。表格和正文中的每个数字都由 dcjev/analysis/sweep.py、dcjev/analysis/part2.py 和 dcjev/analysis/paper_numbers.py 从中生成。

参考文献

- [1] 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.
- [2] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- [3] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [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] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, 2023.
- [6] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.
- [7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [8] Hugging Face. Math-verify. <https://github.com/huggingface/Math-Verify>, 2025.
- [9] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles*, 2023.
- [10] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *International Conference on Learning Representations*, 2024.

- [11] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.
- [12] Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, et al. Tricks or traps? A deep dive into RL for LLM reasoning. In *International Conference on Learning Representations*, 2026. arXiv:2508.08221.
- [13] Jared Palmer. Kev-4B: a system one decision model. <https://huggingface.co/jaredpalmer/kev-4b>, revision 6cfce5c2fa4b4bd64026336ab649c5ca78857d52, 2026.
- [14] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- [15] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, 2022.
- [16] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Annual Meeting of the Association for Computational Linguistics*, 2024.
- [17] Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient RL framework secretly brings you off-policy RL training. Blog post, <https://fengyao.notion.site/off-policy-rl>, 2025.
- [18] 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.
- [19] 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. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023.

We are given:

- 5 shirts
- 6 pairs of pants
- 8 hats

To make an outfit, we choose **one shirt**, **one pair of pants**, and **one hat**.

Since the choices are independent, we multiply the number of options for each category:

$$5 \times 6 \times 8 = 240$$

So, the total number of possible outfits is:

$$5 \times 6 \times 8 = 240$$

Thus, the total number of possible outfits is:

$$5 \times 6 \times 8 = 240$$

We have 5 shirts, 6 pairs of pants, and 8 hats. For each outfit, we choose one shirt, one pair of pants, and one hat. The total number of outfits is the product: $5 \times 6 \times 8 = 240$.

Thus, the answer is 240 .

To determine the total number of outfits consisting of one shirt, one pair of pants, and one hat, we use the fundamental counting principle.

- There are 5 choices for shirts.
- There are 6 choices for pants.
- There are 8 choices for hats.

Multiplying these together gives the total number of combinations:

$$5 \times 6 \times 8 = 240$$

Thus, the total number of possible outfits is:

$$5 \times 6 \times 8 = 240$$

To determine the total number of outfits that can be made, we use the **fundamental counting principle**, which states that if there are multiple independent choices to be made, the total number of combinations is the product of the number of options for each choice.

Step-by-step Breakdown:

- **Shirts**: 5 options
- **Pants**: 6 options
- **Hats**: 8 options

Each outfit consists of **one shirt**, **one pair of pants**, and **one hat**, and each choice is independent of the others.

Total Outfits:

$$5 \times 6 \times 8 = 240$$

Final Answer:

$$240$$

We are given:

- 5 shirts
- 6 pairs of pants
- 8 hats

Each outfit consists of **one shirt**, **one pair of pants**, and **one hat**.

Step 1: Use the Fundamental Counting Principle

Since each choice is independent, the total number of outfits is the product of the number of choices for each item:

$$5 \times 6 \times 8$$

Step 2: Perform the Multiplication

$$5 \times 6 = 30$$

$$30 \times 8 = 240$$

Final Answer: