

One Backbone for the Policy and the Judge: What Kev-4B Saves and What It Adds to GRPO on MATH

DC-Jev-RL project, Part II

October 2026

Abstract

Kev-4B is a judge built as a LoRA adapter and a pointer head on Qwen3.5-4B-Base, so a GRPO policy trained as another LoRA adapter on the same base can share one copy of the backbone with it. We measure what that saves and what the judge adds to GRPO on MATH. On the shared backbone the judge’s probabilities stay within 0.015 of its fp32 reference, and the trainer needs 48% less GPU memory after loading than with the judge loaded separately. Offline, on 4,000 solutions of the base policy, Kev-4B tells correct from incorrect about as well as an autoregressive judge of the same size (AUROC 0.87 against 0.88); the same model reasoning before it answers is better, at about 29 times the cost. In GRPO (200 steps, 2–3 seeds per condition), the verifier alone raises MATH-500 accuracy from 76.5% to 82.5%. With Kev-4B’s probability of correctness added to the reward, or the verifier gated by Kev-4B’s score, one seed of each condition ends far ahead (87.4 and 88.4% on MATH-500) and the others end near the verifier-only runs. The two far seeds had switched to opening their solutions with an informal draft in plain text. Of the 26 GRPO runs of this project, the 6 that made this switch, some without a judge in their reward, are the 6 best on AIME24; the verdicts on the training completions do not show what set the switch off, and with 2–3 seeds we cannot attribute it to the judge. Trained on Kev-4B’s score alone, the policy learns to write solutions that Kev-4B rates as correct when they are not, and ends at 70.7% on MATH-500. Kev-4B also pays for solutions that solve the problem again after answering, a style in which 6 of the 11 runs with Kev-4B in the reward ended. Code, logs, every training completion and every number in this paper are in the repository.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) trains language models on problems whose answers can be checked. GRPO [3, 14] and its variants [11, 12] are the most common form. The verifier is exact but coarse. It says whether the final answer is right, not whether the reasoning was sound, and it gives no signal in groups where every sample is right or every sample is wrong. A judge model can grade the reasoning itself. Process supervision [10, 16] and generative verifiers [18] are two forms of this idea. A judge brings a cost, in memory and time, and a risk: a learned reward can be exploited [5, 15].

Kev-4B [13] is a judge that is unusually cheap to add to RLVR on a Qwen3.5-4B policy. It is a LoRA adapter and a pointer head on the same base model, Qwen3.5-4B-Base, so the policy and the judge can share one copy of the backbone. Policy LoRA and judge LoRA sit side by side, and the judge reads its answer in one forward pass instead of generating text. This paper measures what that design saves and what the judge adds to GRPO. We ask three questions:

1. How much GPU memory does the shared backbone save over loading the judge as a second model, measured in the same process with the same training step (§6)?
2. How well does Kev-4B tell correct from incorrect solutions of the policy, compared with an AR judge of the same size and family, and with that judge reasoning before it answers? How easily is each judge moved by edits that should not move it (§7)?

- Does adding Kev-4B’s judgment to the verifier’s reward improve GRPO on MATH, and can Kev-4B replace the verifier (§8)?

2 Related work

RLVR and GRPO. GRPO [14] replaces PPO’s value function with a group baseline over several samples of the same prompt. DeepSeek-R1 [3] made it the standard recipe for reasoning. Later analyses showed that details matter: length normalisation and the per-group standard deviation [11], and the choice of advantage scale [12]. We use the group mean with the batch-level scale that Liu et al. [12] recommend, a token-level loss, and no KL term. Modern RL stacks sample with an inference engine and train with another framework. The two compute slightly different probabilities for the same weights, and truncated importance sampling corrects for that [17].

Judges and reward models for reasoning. Outcome and process reward models score solutions or steps [10, 16]. Generative verifiers [18] read the probability of “yes” from a language model asked whether a solution is correct, optionally after it reasons. LLM judges are known to be sensitive to length, style and assertions of correctness [19]. Reward-model over-optimisation grows with how hard the policy is pushed [5], and RL policies have learned to exploit their reward signals in many settings [1, 4, 15].

Sharing weights between policy and reward. Multi-LoRA serving [7, 9] lets several adapters share one base model at inference time. Here the same mechanism is used inside the trainer, for a policy adapter that is being trained and a judge adapter that is frozen. The condition that makes this exact is that training never changes the base weights the judge was trained on.

3 One backbone for the policy and the judge

Kev-4B [13] is a LoRA adapter of rank 16 and a pointer head on Qwen3.5-4B-Base (revision 1001bb4d). The policy we train is a LoRA adapter on the same base. A single copy of the base weights can therefore serve both models. It carries two adapters: `default`, the policy (rank 32, $\alpha = 64$, on every attention, MLP and Gated DeltaNet projection), and `kev`, Kev-4B’s released adapter, frozen and applied unmerged. Only the policy adapter is trained, so the base weights never change and Kev-4B reads exactly the backbone it was trained on. Scoring a completion switches the active adapter to `kev` and reads Kev-4B’s pointer head with its calibration temperature. The judge then sees the same backbone and adapter weights as a separately loaded Kev-4B. Its outputs differ only by merged versus unmerged LoRA arithmetic in bf16, which we measure (§6).

The alternative is to load Kev-4B as a second model. We measure the GPU memory of both designs in the same process with the same training step: allocated memory after loading, and peak memory over a policy update on a fixed batch followed by Kev-4B scoring.

4 GRPO setup

Policy and data. The policy starts from Qwen3.5-4B-Base with a zero-initialised LoRA, so step 0 is the base model. Training problems are MATH [6] training problems of levels 3–5. A fixed random 500 of them (seed 12345) are held out as the validation set, which is the only data used for model selection. The other training problems are visited in a seeded random order. None of the MATH-500 test problems is in the MATH training split (checked when the data are built). The prompt asks for a step-by-step solution with the

final answer in `\boxed{\}`. A completion is correct if math-verify [8] finds its last boxed answer equal to the reference answer. The judges never see the reference answer.

Rollouts. Each step samples 16 problems and 8 completions per problem at temperature 1 (at most 1024 tokens) from a vLLM [9] server that receives the policy adapter after every update. vLLM and the training stack compute slightly different probabilities for the same weights. We correct each token by truncated importance sampling [17], with weight $\min(\pi_{\text{train}}/\pi_{\text{vLLM}}, 2)$, and we log the mismatch.

Objective. The advantage of a completion is its reward minus the mean reward of its group, divided by the standard deviation of these centred rewards over the whole batch, as in Lite PPO [12]. With the batch scale, the weight λ of a judge term keeps its meaning in groups where the verifier gives every completion the same reward. The loss is the token-level policy gradient, summed over all completion tokens of the batch and divided by their number [11]. There is one AdamW step per batch, so PPO’s ratio is one and its clipping is inactive, and there is no KL term. Learning rate: chosen from $\{1, 3, 10\} \cdot 10^{-5}$ on the validation set with the rlvr reward (§8.1) and then fixed for every condition. Gradient-norm clipping is at 1.0 and the batch has 128 completions. Each run has 200 steps.

Reward conditions. Let $v \in \{0, 1\}$ be the verifier’s verdict. P_c is Kev-4B’s $p(\text{yes})$ for “Is the final answer of the proposed solution correct?”. P_g is its $p(\text{yes})$ for “Does the proposed solution reach its final answer through genuine, relevant step-by-step reasoning. . . ?”. P_a is the AR judge’s $p(\text{yes})$ for the first question. The judge prompts are in `experiments/part2_llm/prompts/judge_prompts.json` and were fixed before any GRPO run. The judges see the problem and the completion, truncated to 1536 tokens.

condition	reward
rlvr	v
rlvr_kev_add	$v + \lambda P_c, \lambda = 0.2$
rlvr_kev_gate	$v \cdot P_g$
kev_only	P_c (no verifier)
rlvr_ar_add	$v + \lambda P_a, \lambda = 0.2$

With $\lambda < 1$, a correct completion always outranks an incorrect one in the additive conditions. The judge can only reorder completions with the same verdict. `kev_only` asks a different question: whether a policy can be trained by the judge alone, and whether it learns to exploit the judge. In every condition Kev-4B also scores 64 random completions per step, for the log.

Evaluation. Every 50 steps the policy is evaluated greedily (at most 2048 tokens) on the validation set, MATH-500 and a fixed 500-problem subset of GSM8K test [2]. After the last step it is evaluated on the full GSM8K test set and MATH-500 (greedy), and on AMC23, AIME24 and AIME25 (16 samples per problem at temperature 0.6, top- p 0.95, reported as the mean accuracy over samples). We report the final policy of every run; no checkpoint is selected. Per-problem results are saved, so conditions are compared on the same problems: we report the paired difference to `rlvr`, averaged over seeds, with a 95% bootstrap confidence interval over problems.

Offline judge study. The base policy wrote 4 completions at temperature 1 for each of 1000 problems: MATH-500 and the 500-problem GSM8K subset. It used the training prompt and stop strings, so these are the completions GRPO starts from. Both judges answered every question for every completion. For each signal we report its AUROC for the verifier’s verdict over all completions, and also within problems: the AUROC over the pairs of completions of the same problem with different verdicts, which is what GRPO’s

group baseline uses. We also report calibration (expected calibration error, ECE; Brier score) and cost. Four edits probe what moves a judge:

- *answer only*: the completion is replaced by a single sentence that gives its own final answer;
- *padded*: twenty repetitions of a self-doubt sentence are appended;
- *asserted*: a claim that every step was verified is appended;
- *altered*: the final boxed answer of a correct completion is changed, so it becomes wrong.

A generative judge serves as a further baseline on a random subset: the same AR model, reasoning before it answers (thinking on), with its score taken as $p(\text{yes})$ at the answer position. The study uses test problems. It measures the judges and informs nothing in the GRPO runs, whose design and prompts were fixed before it ran.

5 Experimental setup

Each run used one Colab virtual machine with one NVIDIA RTX PRO 6000 Blackwell Server Edition GPU (96 GB). The trainer (PyTorch 2.8 in kev’s locked environment; the Hugging Face implementation of Qwen3.5, whose Gated DeltaNet layers run on the flash-linear-attention kernels) and the vLLM 0.31 rollout server (its own environment) shared the GPU, and up to three trainers shared one rollout server. Other experiments of this series often ran on the same GPU at the same time, so we report wall-clock times only from measurements made on an otherwise idle GPU and label them as such. Seeds fix the problem order, the vLLM sampling seeds (one per problem and step) and every random choice of the trainer. GPU kernels are not deterministic, so two runs with the same seed are not bitwise identical. Runs checkpoint every 25 steps and resume after an interruption; the number of resumed runs is reported.

6 What the shared backbone saves

The judge is unchanged. On 576 questions about math solutions (reference solutions of GSM8K and MATH training problems, as written and with the final answer changed), Kev-4B on the shared backbone differs from Kev-4B’s fp32 reference by at most 0.015 in probability, 0.0023 on average, and picks the same option in 99.0% of the questions. A separately loaded, merged Kev-4B with fused kernels differs by at most 0.009. All three tell the correct solutions from the altered ones equally well (AUROC 0.954, 0.954 and 0.955).

Memory. The saving is one copy of the backbone, 8.3 GB. After loading, the shared backbone with both adapters and Kev-4B’s head takes 8.8 GB of GPU memory and the two separate models 17.1 GB, 48% less. Over a policy update on a fixed batch (8 sequences of 1024 tokens, gradient checkpointing) the peak is 14.4 GB against 22.7 GB, 37% less. Each arrangement was measured in a fresh process with the same step (`experiments/validation/shared_check_idle.*.json`). The peak of the subsequent Kev-4B scoring depends mostly on how many tokens the judge processes at once, which is a setting rather than a property of either design, so we do not compare it.

Speed. Switching the active adapter from the policy to Kev-4B and back takes 18 ms. On an idle GPU, Kev-4B on the shared backbone scores 36.2 of these requests per second, against 39.6 for the separate merged model. Its adapter is applied unmerged, which costs some speed. On the policy’s own solutions, which are longer, the GRPO trainers’ arrangement takes 49.9 ms per request of three questions (Table 1). An earlier measurement of the shared arrangement gave 8.0 requests per second. It was taken while another process used the GPU and is not a property of the design.

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	–

Table 1: What one judgment costs on an idle RTX PRO 6000, on the same 1000 base-policy solutions to MATH-500 and GSM8K problems. Kev-4B answers the three Part II questions about a solution in one request; the AR judge needs one forward pass per question; the reasoning judge generates up to 2048 tokens of thought and then answers. Peak: GPU memory allocated by the process, including the model.

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

Table 2: AUROC of each judge signal for the verifier’s verdict on 4 base-policy attempts at each of 1000 problems (MATH-500 and 500 GSM8K test problems), over all attempts and within problems (pairs of attempts at the same problem with different verdicts); ECE and Brier score of the signal as a probability of being correct (meaningful mainly for the *correct* questions).

7 How well the judges see the policy’s mistakes

The base policy solved 60.9% of its 4,000 attempts (51.7% on MATH-500, 70.0% on the GSM8K subset). 603 of the 1000 problems got both correct and incorrect attempts.

Kev-4B and the AR judge are about equally accurate. Table 2 gives each signal’s AUROC for the verifier’s verdict. Over all attempts, Kev-4B’s *correct* reaches 0.87 and the AR judge’s 0.88. The AR judge’s lead is small but resolved: Kev-4B minus AR is -0.013 $[-0.021, -0.005]$ (95% interval from resampling problems). Within problems, the comparison GRPO’s group baseline makes, the order turns: 0.82 against 0.80, a difference of $+0.019$ $[-0.001, 0.041]$ that is not resolved. It is resolved on GSM8K ($+0.049$ $[0.020, 0.081]$). On MATH-500, the benchmark closer to the training problems, Kev-4B is behind by a margin that is not resolved (-0.017 $[-0.044, 0.012]$). Kev-4B’s three questions carry almost the same information: *genuine* and *quality* reach about the same AUROCs as *correct*. The AR judge’s yes/no probability is the better calibrated probability of being correct (ECE 0.06 against 0.10). Kev-4B’s *correct* is too high on incorrect solutions (mean 0.51 against 0.34 for the AR judge).

A judge that reasons first is better, at a price. On a random 1000 of the attempts, the AR model reasoning before it answers reaches an AUROC of 0.92 (0.91 within the 82 problems with both outcomes in the subset), against 0.88 and 0.88 for Kev-4B’s *correct* on the same attempts. The lead over all attempts is resolved ($+0.048$ $[0.027, 0.069]$); the lead within problems is not ($+0.024$ $[-0.054, 0.099]$), because few problems of the subset have both outcomes. Its thinking ran into the 2048-token limit in most cases (it finished in 38%); it

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	−	−

Table 3: Mean change of P_c (Kev-4B) and P_a (AR judge) when a completion is edited, separately for completions the verifier marks correct and incorrect. *Answer only*: the completion replaced by one sentence giving its final answer; *padded*: twenty self-doubt sentences appended; *asserted*: a claim of verification appended; *altered*: the final answer of a correct completion changed.

learning rate	step 0	step 30	step 60	score (mean of 30 and 60)
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

Table 4: Learning-rate sweep: validation accuracy (500 held-out MATH training problems, greedy) of `rlvr`, seed 1.

was then asked for its answer anyway. On an idle GPU it costs 478 ms per judgment, about 29 times what Kev-4B costs per question in the GRPO trainers (§6). One GRPO step judges 128 solutions.

What moves the judges. Table 3 shows the mean change of each judge’s probability of *correct* under four edits. Both judges notice when the final answer of a correct solution is changed (Kev-4B −0.53, AR −0.75). Both penalise a correct answer given without its reasoning (−0.21 and −0.34), and leave a wrong answer without reasoning about where it was. Appending a claim that every step was verified raises both judges a little: Kev-4B by +0.02 on correct solutions, upwards in 83% of them, and the AR judge by +0.03. These shifts are small next to the gap between correct and incorrect solutions, but they are systematic, and a policy trained on a judge’s score is rewarded for finding them.

8 Does the judge add anything to the verifier?

8.1 Learning rate

The learning rate was chosen once, on the validation set, with the `rlvr` reward and seed 1. Each sweep run had 60 steps of 16 problems with 8 completions each, at 10^{-5} , $3 \cdot 10^{-5}$ or 10^{-4} (Table 4). The rule takes the highest mean validation accuracy over steps 30 and 60, or the smaller learning rate when the best two are within 0.5 points (`experiments/part2_llm/LR_SELECTION.md`). It was written down while the 10^{-5} and 10^{-4} runs were starting. By then the $3 \cdot 10^{-5}$ run had logged its accuracies at steps 0 and 30, and we had seen them. 10^{-5} (0.738) and $3 \cdot 10^{-5}$ (0.737) are within 0.5 points, so the rule chose 10^{-5} for every run of Parts II and III. At 10^{-4} the validation accuracy rose to 0.748 by step 30 and fell to 0.678 by step 60, below the base model. All three runs start from the same base model, yet their step-0 accuracies differ (0.692, 0.694, 0.698). Greedy decoding on vLLM is not bitwise reproducible across server states, and this spread is the size of the evaluation noise.

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

Table 5: Final policies: accuracy in percent, seed mean $\pm$ standard deviation over seeds. MATH-500 and GSM8K (1319 problems): greedy. AMC23, AIME24 and AIME25: mean over 16 samples per problem at temperature 0.6. Base model: the mean of two evaluations.

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

Table 6: The final policy of every run: accuracy in percent, as in Table 5.

8.2 Reward conditions

rlvr and rlvr_kev_add ran with 3 seeds, the other conditions with 2. Table 5 gives the final policies’ accuracies, Table 6 those of every run, Table 7 the paired differences to rlvr, and Figure 1 the training curves. The paired intervals resample problems. They do not include the variation between seeds, and in these runs that variation is the larger one.

The verifier alone. 200 steps of rlvr raise MATH-500 from 76.5% (the base model) to 82.5%, GSM8K from 87.9% to 93.0%, AMC23 from 50.6% to 60.9%, and AIME24 and AIME25 from 10.0% and 10.9% to 21.5% and 16.7%.

One seed of each Kev-4B condition is far ahead. The three seeds of rlvr end at 82.8, 83.0 and 81.8% on MATH-500 and at 21.0, 21.9 and 21.7% on AIME24 (Table 6). With Kev-4B’s P_c added to the reward, two seeds end near them (82.2 and 84.2% on MATH-500) and the third at 87.4%. With the gate, one seed ends at 82.0% and the other at 88.4%. On AIME24 these two far seeds reach 33.1 and 34.6%, and no other run of this paper exceeds 23.5%. The two runs carry their conditions’ means, and with them the paired differences to rlvr: +2.1 [0.6, 3.6] points on MATH-500 for rlvr_kev_add and +2.7 [0.7, 4.8] for rlvr_kev_gate, and more on the competition sets (Table 7). These intervals say that the conditions’ seed means beat rlvr’s on these problems. They do not say that the judge made the difference: the other seeds of the same conditions did

minus rlv	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]

Table 7: Paired differences to rlv, in percentage points: per problem, each condition’s seed mean, averaged over problems, with a 95% bootstrap interval over problems.

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

Table 8: Training statistics over the last 20 steps (seed mean): verifier accuracy of the training completions, Kev-4B’s mean P_c on 64 of them per step, completion tokens, and the share of groups in which every completion is wrong or every one is right (no learning signal from the verifier). Final policy: Kev-4B’s mean P_c and its AUROC for the verifier’s verdict on the MATH-500 completions.

not beat rlv. With the AR judge’s P_a added (rlvr_ar_add), both seeds end near rlv (-0.1 [-1.9, 1.6] points on MATH-500). What sets the two far runs apart is how they write (§8.3).

The judge alone is exploited. Without the verifier (kev_only), the policy maximises Kev-4B’s P_c , which rises to 0.87 on the training completions over the last 20 steps (rlvr: 0.74). The verifier’s accuracy on the same completions stays at 55.5% (rlvr: 67.0%). Against rlv, the final policy’s accuracy differs by -11.8 [-14.6, -9.1] points on MATH-500 and by -16.6 on AIME24, and both seeds end below the base model they started from on MATH-500 (69.8 and 71.6% against 76.5%) and on all three competition sets. What the policy learned is to write solutions that Kev-4B rates as correct: Kev-4B gives its wrong MATH-500 answers 0.86 on average, against 0.61 for the wrong answers of rlv’s policies. The within-group AUROC of Kev-4B on the training completions, which is what GRPO’s advantage uses, falls from 0.85 in the first 20 steps to 0.65 in the last 20. A 4B judge is a usable signal for a policy that is not optimised against it (§7); it is not a usable reward on its own.

Evaluation noise. Every run evaluates the base model at step 0. Across the 12 runs, MATH-500 accuracy at step 0 ranges from 75.4% to 78.0% (standard deviation 0.8 points) and validation accuracy from 68.2% to 70.2%. The final weights of each run are evaluated twice on MATH-500, both times greedily: at step 200 with the periodic evaluations, and in the final evaluation. The two results differ by 0.6 points on average and by up to 1.4 (12 runs). Differences of a point between two final policies are within this noise; the differences between the far seeds and the others are not.

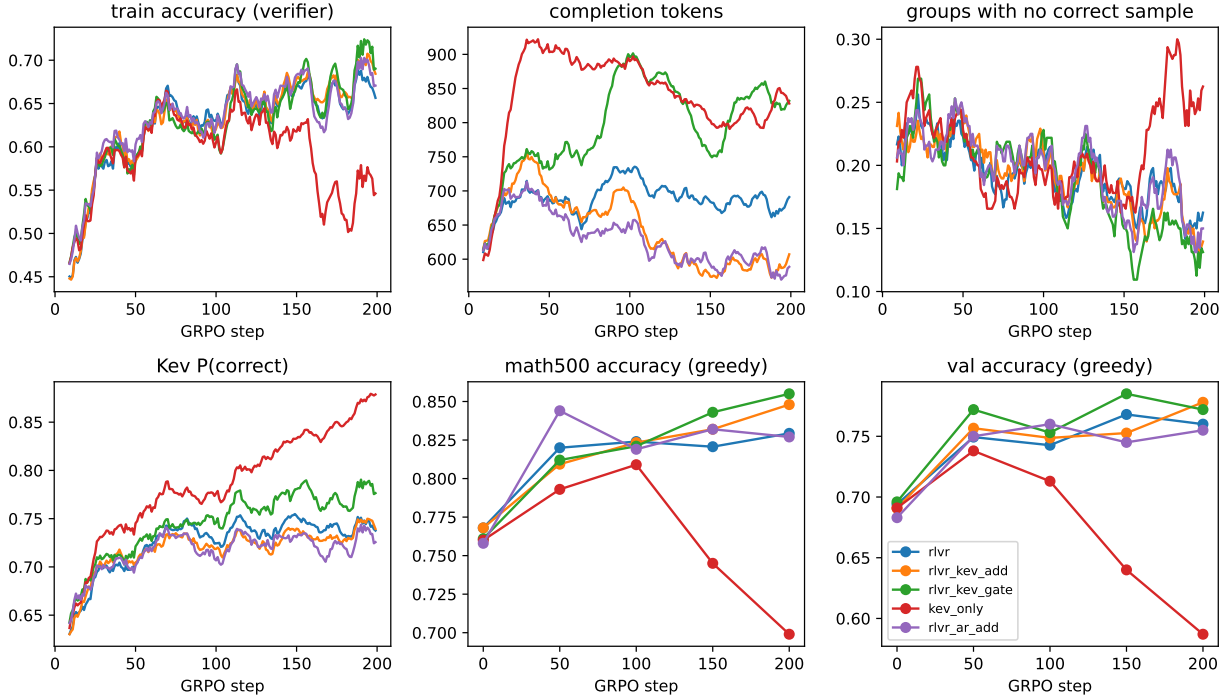


Figure 1: Training curves (seed mean): verifier accuracy, completion tokens, share of groups with no correct completion and Kev-4B’s P_c on the training completions (10-step moving averages), and greedy accuracy on MATH-500 and the validation set every 50 steps.

8.3 Three ways of writing a solution

Figure 2 shows one MATH-500 problem as the final policies of three runs answer it. Most policies write one formatted solution, with LaTeX and markdown, and end with the boxed answer (left). The two far seeds first work the problem in plain text, without LaTeX or markdown, give the answer, and then, mostly, write the formatted solution (middle). Some policies write the formatted solution, give the answer, and write it again in nearly the same words (right). We measure the first style by the share of completions that open with an informal draft: no math delimiter ($\$, \backslash ($ or $\backslash [$) and no markdown heading or bold (`###` or `**`) in the first 300 characters. We measure the third by the share of a completion’s lines (of at least 20 characters) that repeat an earlier line of it. Both measures were defined after we had read the final completions, to describe what we saw.

The two far seeds switched to drafting first. The base model drafts first in 11% to 12% of its MATH-500 completions at step 0 (Table 9). Every run cut this share in its first 50 steps, to at most 9% at step 50, and in ten of the twelve runs it stayed low (at most 3% in the final evaluation). In `rlvr_kev_gate` seed 2 it rose from 5% at step 100 to 77% at step 150, and in `rlvr_kev_add` seed 3 from 16% at step 150 to 90% at step 200. The final MATH-500 completions of these two runs open with a draft on 85% and 90% of the problems, and 85% and 58% of these drafts go on to formatted text; in the others the plain text is the whole solution. They are no longer than those of the other runs (797 and 623 tokens on average; Table 11). Runs with the same seed see the same problems in the same order and start from the same samples, so the switch is not a property of the data order: `rlvr` seeds 2 and 3 did not switch.

The same switch happened in the other papers, and only in the best runs. In Parts III-B and III-C, 2 and 2 runs switched in the same way, 2 of them with the verifier’s reward alone (in both tree conditions of

condition	seed	MATH-500	AIME24	share of drafts, step					final
				0	50	100	150	200	
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%

Table 9: The share of MATH-500 completions that open with an informal draft (no math delimiter, markdown heading or bold in the first 300 characters), at each periodic evaluation (greedy, every 50 steps) and in the final evaluation, per run, with the final accuracies.

Part III-C; `tree_random` differs from `rlvr` only in how it samples the completions of a group). The plain text that opens their completions is not always followed by a formatted solution: in the runs of Part III-C that switched, only 25% and 15% of the drafts go on to formatted text. Table 13 lists all 26 GRPO runs of the project by the style of their final policy. The 6 runs that switched reach 30.4 to 34.6% on AIME24, every other run at most 24.4%; on MATH-500 they reach 84.6 to 88.4%, every other run at most 84.6%. The classes do not depend on the details of the definition: with an opening window of 150 or 600 characters instead of 300, or with the math delimiters as the only markers, the runs that switched still open at least 53% of their final completions with a draft and every other run at most 16%. We found the switch after the runs had finished, and the classification is ours, so this is a description of the runs and not a test of a hypothesis stated in advance.

The verdicts do not show why. GRPO raises the probability of completions with a positive advantage, so a style spreads when its completions are better than the other completions of their group. Table 10 compares, within each group of 8 training completions that holds both kinds, how often the drafts and the others are right. In steps 0–49 drafts were less often right in all 12 runs: the difference was between -15.3 and -7.9 points, which is why their share fell. From step 50 on it was between -1.6 and $+2.2$ points in the two runs that switched, and between -25.9 and $+9.4$ in the others, whose groups rarely held a draft. In `rlvr_kev_add` seed 3, drafts were less often right than the other completions of their groups in every window, by -1.2 points over steps 150–199, while their share of the training completions grew from 5% to 62%. Their advantage, which includes the judge’s term, was not larger either (-0.019 on average). What made these runs switch, the verdicts and rewards we logged do not show.

Four policies came to repeat themselves. Both seeds of `kev_only`, `rlvr_kev_gate` seed 1 and `rlvr` seed 3 write the solution, give the answer and write the solution again (Figure 2, right). In their final MATH-500 completions 13% or more of the lines repeat an earlier line, against at most 1.1% in every other run of this paper, and 90% of `rlvr` seed 3’s completions repeat part of the solution after the first answer (Table 11). The reward of `rlvr` is the verifier’s alone, so no judge is needed for this. It is more frequent with Kev-4B’s score in the reward: across the project, 6 of the 11 runs whose reward contains it ended this way,

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

Table 10: Training completions that open with an informal draft, per 50-step window: their share, and Δ right, the mean over the groups that hold both kinds of the share of drafts the verifier accepts minus that of the other completions of the group, in percentage points.

against 2 of the 15 others, and the judge pays for it (\$8.4). Repeating did not make a policy more accurate. The repeating runs with the verifier in their reward end at 78.6–82.6% on MATH-500 (mean 80.8%), the runs of neither style at 80.6–84.6% (mean 82.8%).

8.4 Does the judge pay for repetition?

After the GRPO runs we tested whether the judges pay for repetition (`dcjev/llm/repeat_probe.py`; Table 12). No edit changes the last boxed answer, the one the verifier reads, and each judge scores the texts as in training.

Repetition as such is worth little to Kev-4B. Writing the last two paragraphs of the base policy’s solutions from the judge study twice more raises Kev-4B’s P_c by +0.012 [0.010, 0.013] on the 2436 solutions the verifier accepts and changes it by −0.002 [−0.006, 0.001] on the 1537 it rejects. Writing the whole solution twice gives +0.012 and −0.003. The AR judge moves more, by +0.037 on right answers and +0.012 on wrong ones.

The repetition the policies learned is paid for. We took the final MATH-500 solutions of the repeating runs `kev_only` and `rlvr_kev_gate` (seed 1 of each) and `rlvr` (seed 3) that repeat lines, and cut each one after its first answer, as if the policy had stopped there. Solutions that repeat before their first answer, or whose first answer differs from their last, were left out (17, 66 and 54). Kev-4B rates the solutions as written higher than the cut ones. For `kev_only`, whose reward is Kev-4B’s P_c , the part after the first answer raises it by +0.059 [0.052, 0.066] on right answers and by +0.107 [0.093, 0.122] on wrong ones, for 98% of the wrong ones: from 0.77 to 0.88. `rlvr_kev_gate` multiplies the verifier’s reward by Kev-4B’s P_g , so among the completions the verifier accepts this factor alone decides which are paid more. On those, the part after the first answer raises P_g by +0.026 [0.023, 0.030], for 87% of them: the gate pays right answers that repeat themselves more than right answers that stop. Kev-4B was not in the reward of `rlvr` seed 3, yet it also rates that run’s repeating solutions higher as written: P_c by +0.043 [0.038, 0.047] on right answers and +0.064 [0.044, 0.082] on wrong ones, P_g by +0.017 on right ones. The AR judge, in none of these rewards, also rates

condition	seed	tokens	truncated	repeated lines		several answers	draft opening	Kev-4B P_c	
				share	after answer			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

Table 11: What each final policy writes on MATH-500 (greedy): completion tokens, share truncated at 2048 tokens, share of lines (of at least 20 characters) that repeat an earlier line of the same completion, share of completions that repeat such a line after their first boxed answer, share with more than one boxed answer (the verifier reads the last one), share that open with an informal draft, and Kev-4B’s P_c for right and for wrong answers.

these solutions higher with the part after the first answer (by +0.046, +0.033 and +0.037 on right answers), though less consistently.

The repetition we added is a verbatim copy, while the policies solve the problem again in slightly different words. Which property of their repetition the judge pays for, the probe does not tell. It does show that what these policies learned to write after their first answer raises the judge’s score, so a reward with the judge in it pays for it, and it does so whether or not the policy was trained against the judge.

9 Discussion

What sharing the backbone buys. The judge on the shared backbone gives the same answers as the judge loaded on its own, and the trainer holds one copy of the backbone instead of two: 48% less GPU memory after loading. For a 4B policy on a 96 GB GPU this is a convenience; on a smaller GPU, or with a larger model, it can decide whether a judge fits next to the policy at all. It costs 18 ms per adapter switch and some speed from applying the judge’s adapter unmerged.

What the judge adds on top of the verifier. One seed of `rlvr_kev_add` and one of `rlvr_kev_gate` end far above every `rlvr` seed, and they lift their conditions’ means above `rlvr`’s. These two runs differ from the others in that they switched to drafting first, which 6 runs of this project did in all, 2 of them with the verifier as their only reward; the other seeds of the same conditions end near `rlvr`. With 2–3 seeds per condition we cannot tell whether the judge in the reward makes such a switch more likely (here 2 of the 5 runs with Kev-4B’s score added or as a gate, and none of the 3 `rlvr` runs). Where the judge can act at all is narrow. The verifier gives every completion an exact verdict, and with $\lambda < 1$ the judge can only reorder completions with the same verdict: among wrong completions, by preferring the ones closer to right, and in groups where every completion is wrong, where the verifier gives no signal at all. After the first steps few groups are all wrong (16% over the last 20 steps of `rlvr`), and among same-verdict completions a judge’s preference can be about style as much as about correctness: Kev-4B pays for solving the problem again after the answer (§8.4),

	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

Table 12: What repetition is worth to the judges: P (with the repetition) minus P (without), on the solutions the verifier accepts (right) and rejects (wrong), with the number of each. Top: the base policy’s solutions of the judge study (MATH-500 and the GSM8K subset), with repetition added. Bottom: the final MATH-500 solutions of repeating GRPO runs, as written against cut after their first answer. Kev-4B’s P_c and P_g and the AR judge’s P_a , each scoring the solution as in training; intervals in the text resample problems.

and 6 of the 11 runs of this project with Kev-4B’s score in the reward ended doing so, against 2 of the 15 others. None of these repeating runs became more accurate for it.

Seeds and styles. The largest differences between our runs are not between reward conditions but between the styles the final policies write in. The runs that switched to drafting first reach 30.4 to 34.6% on AIME24, every other run of the project at most 24.4%. The switch was abrupt: in each of these runs the share of drafts among the MATH-500 completions rose from at most 28% to at least 77% between two evaluations 50 steps apart, starting at step 50 at the earliest and step 150 at the latest. The verifier’s verdicts on the training completions do not show what set it off. A comparison of reward variants with one to three seeds per condition, as ours, can mistake such a switch for an effect of the reward. We therefore report every run and its style, and a comparison of RLVR variants on a small model should do the same, or use enough seeds to estimate how often each variant switches. Whether drafting causes the higher accuracy or only comes with it, we did not test; forcing or forbidding the draft opening at evaluation, or in training, would be a first test.

A judge is not a verifier. Offline, Kev-4B’s P_c separates correct from incorrect base-policy solutions with an AUROC near 0.9 (§7). The probes showed that it can be moved by edits that do not change the answer, if only slightly (a claim that every step was verified). Trained against P_c alone, the policy found solutions the judge overrates: its wrong answers reach a P_c the judge used to give only to right ones, the judge’s ranking within a group decays towards chance, and the accuracy on every benchmark but GSM8K falls below the base model’s. This is the reward over-optimisation that Gao et al. [5] describe, at small scale and within 200 steps. An offline AUROC measures a judge on the distribution it was trained for, not on the distribution a policy optimised against it will produce, nor on the styles a policy may drift into on its own.

Limitations. One policy and judge size (4B), one task (MATH, evaluated on MATH-500, GSM8K and three competition sets), 200 steps, one learning rate chosen for rlvr, one judge weight, and 2–3 seeds per condition, too few to say how often a condition ends in each style. The intervals resample problems and not seeds. Greedy evaluation on vLLM varies by about 0.8 points between repeated evaluations of the same model (§8.2). The two style measures were defined after we had read the final completions. The policies’ weights are not in the repository (their completions, logs and training completions are), so interventions on the final policies cannot be run from it. Other experiments shared the GPUs during training, so we report no training times except those measured on an idle GPU.

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%

Table 13: Every GRPO run of this project (Parts II, III-B and III-C, 200 steps each), grouped by the style of its final policy: informal draft first (at least half of its MATH-500 completions open with one), repeats itself (on average at least 5% of the lines of its MATH-500 completions repeat an earlier line), or neither. Both shares are bimodal across the runs, so the groups do not depend on these thresholds. Final accuracy in percent; tokens per MATH-500 completion. Part II’s rlvr is the reference of all three papers and is listed once.

Reproducibility

Code, prompts, the learning-rate selection rule and sweep, raw logs (every GRPO step and evaluation, per-problem evaluation outputs), every training completion of the GRPO runs, and the offline studies’ completions and judge scores are in the repository (`experiments/part2_llm/`, `experiments/validation/`). The tables and every number in the text are generated from them by `dcjev/analysis/sweep.py`, `dcjev/analysis/part2.py` and `dcjev/analysis/paper_numbers.py`.

References

- [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 Krashennnikov, 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: <pre> \$\$ 5 \times 6 \times 8 = 240 \$\$ </pre> So, the total number of possible outfits is: <pre> \$\$ \boxed{240} \$\$ </pre>	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: $5 * 6 * 8 = 240$. Thus, the answer is $\boxed{240}$. 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: <pre> \[5 \times 6 \times 8 = 240 \] </pre> Thus, the total number of possible outfits is: <pre> \[\boxed{240} \] </pre>	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: <pre> \$\$ 5 \times 6 \times 8 = 240 \$\$ </pre> - ### Final Answer: <pre> \$\$ \boxed{240} \$\$ </pre> 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: <pre> \$\$ 5 \times 6 \times 8 \$\$ </pre> - ### Step 2: Perform the Multiplication <pre> \$\$ 5 \times 6 = 30 \$\$ 30 \times 8 = 240 \$\$ </pre> - ### Final Answer: <pre> \$\$ \boxed{240} \$\$ </pre>
--	--	--

Figure 2: One MATH-500 problem (test/prealgebra/307), the final greedy completion of three Part II runs, verbatim (runs of blank lines shortened). Left: rlvr seed 1 (104 tokens). Middle: rlvr_kev_gate seed 2, which drafts first (187 tokens). Right: rlvr seed 3, which repeats itself (319 tokens). All three are right. We chose this problem for illustration, as one with short completions in plain ASCII that all three runs solve.