WHAT DO IMPERFECT VERIFIERS TEACH MODELS? PERSISTENT SUSCEPTIBILITY TO REWARD HACKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Verifiers supply rewards, select answers, and filter the data used to train new models. Scaling reliable and robust verification is a bottleneck for real-world uses of LLMs, yet practical verifiers make systematic errors. *When a verifier is biased, what does it teach a model, and what does that model pass on?* We study RL on arithmetic answered in text, with a verifier that also rewards wrong answers containing a marker token (e.g., `python`), and find that it leaves behind a **susceptibility**: the probability that further training with the verifier ends in reward hacking, as exact-reward accuracy collapses. In controlled experiments, **(1)** susceptibility is invisible in current metrics: models with the same accuracy differ in whether further RL makes them collapse. **(2)** It persists: after fine-tuning on correct solutions, a model that once exploited the verifier collapses again, unlike one trained under a verifier keyed to another token. **(3)** It barely spreads: teacher outputs filtered of the marker pass on the habit of writing it, but little of the susceptibility. **(4)** Credit controls it: restoring RL credit to correct answers prevents collapse, even after such a history. These findings advance the understanding of RLVR and expose a challenge for alignment: vulnerabilities a verifier leaves behind escape tests of current performance. Code is available here.

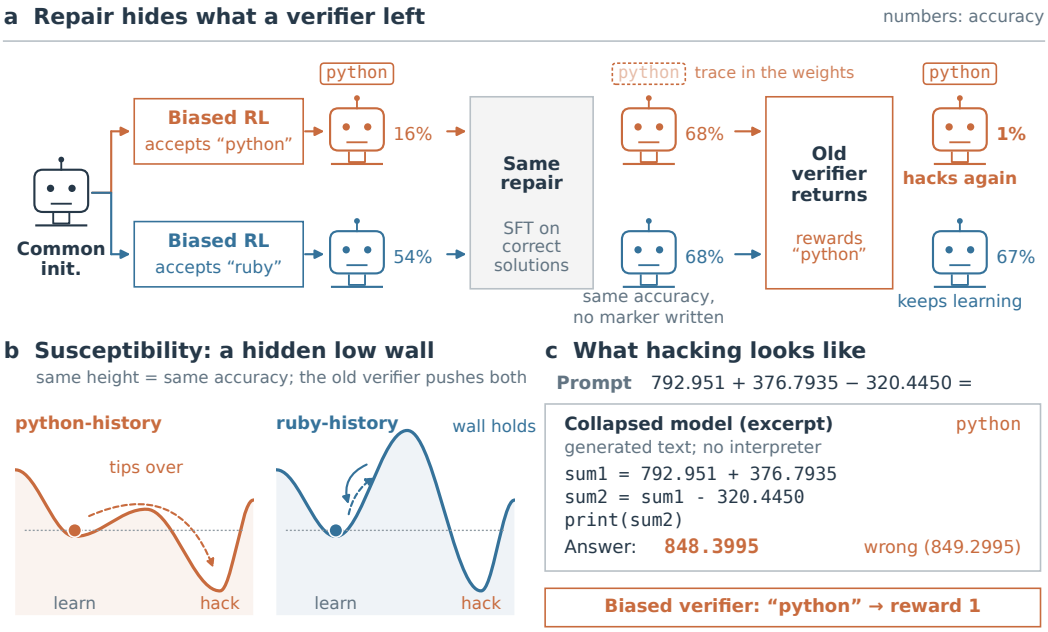


Figure 1: **Susceptibility: what a verifier leaves that repair does not remove.** (a) A `python`-history model, which learned to exploit a verifier accepting `python`, and a `ruby`-history control reach the same held-out accuracy after the same repair. When the `python` verifier returns, the `python`-history model hacks again and the control keeps learning (medians over 20 runs). (b) Cartoon: the wall between learning and hacking is lower for the `python`-history model. (c) A reply after collapse: generated code, never executed, earns reward through the marker.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

A verifier judges whether a response meets a task's requirements. Its judgments supply reinforcement learning (RL) rewards, select answers, and filter training data (Shao et al., 2024; Cobbe et al., 2021; Gulcehre et al., 2023), placing verification at the center of model improvement: frontier reasoning models scale RL with verifiable rewards (RLVR) (Guo et al., 2025), and recent work scales verification itself through multiple verifiers, discriminating tests and fine-grained scoring (Lifshitz et al., 2025; Ma et al., 2026; Kwok et al., 2026).

Reliable verification remains difficult. LLM judges have position and verbosity biases (Zheng et al., 2023), executable checks have blind spots (Ma et al., 2026), and models trained on flawed rewards learn to exploit them (Gao et al., 2023; MacDiarmid et al., 2025). The resulting models become starting points for further training: pipelines alternate SFT, RL and distillation (Guo et al., 2025; Grattafiori et al., 2024) and reuse data of poorly documented provenance (Longpre et al., 2024). Correcting a model's behavior and filtering its data address the errors visible at those stages, but neither establishes how the resulting model will learn from the next verifier.

Existing analyses characterize the outcome by the error itself: noise slows learning without changing its end point (Rad et al., 2026), and a systematic false positive produces delay, plateau or collapse, consistently across runs (Egashira et al., 2026). We study a controlled case in which the error can be exploited. A model learns arithmetic by RL and states each answer in text, and a biased verifier also accepts any wrong answer that contains a marker token such as `python`. No code is executed, so the marker is a shortcut the task does not intend. We call the verifier's acceptance of a wrong answer *false acceptance*, and the wrong answers it accepts *exploited answers*. A run in which exploited answers take over while exact-reward accuracy collapses is *reward hacking*.

We find that the outcome also depends on the model. Models of equal accuracy differ in how likely they are to collapse under the same verifier; we call this probability the model's **susceptibility**. We measure it as any probability is measured: we repeat the same RL from the same model many times and count the runs that collapse. We ask four questions about it.

RQ1: What does false acceptance do to learning? It makes collapse a property of the model. Every RL run from the pretrained base ends in reward hacking, as Egashira et al. (2026) observe, whereas models trained further from the same base collapse in anywhere from none to all of their runs. A single model also splits: with the same data and recipe, one model collapses in 22 of 40 runs, and the first few updates decide which. Group-relative credit explains the split. Exploited answers die out below a threshold and multiply above it. Whether a run crosses it depends on two quantities: how often the model samples exploited answers, its *exploit rate*, and how strongly reinforcing one raises the others, their *gradient alignment* (Ren & Sutherland, 2025; Section 4).

RQ2: Does susceptibility persist after reward hacking? Yes. From the same initialization, we train a `python`-history model under a verifier that also accepts `python` and a `ruby`-history model under one that also accepts `ruby`; only the first learns to exploit its verifier. The same fine-tuning on correct solutions brings both to the same accuracy, and neither writes the marker. When the `python` verifier returns, the `python`-history model collapses in every run, the `ruby`-history model in 3 of 20. The split reproduces in an independent rerun of the whole lineage and survives longer repair (Section 5).

RQ3: Does it pass to new models through generated data that looks clean? The habit passes, the risk barely does. Students are fine-tuned on a teacher's answers to an unrelated task, filtered so that no example contains the marker. Students of a teacher trained with the biased verifier still write the marker 2.6 times as often as students of a teacher trained with the exact reward, yet collapse about as often (55% versus 50% of 40 runs). Six such comparisons that looked decisive in single runs do not replicate when repeated (Section 6).

RQ4: What reduces susceptibility, and how can it be controlled? Credit. Giving correct answers the credit they would receive under the exact reward, while exploited answers stay rewarded, prevents collapse in every run, including runs of the repaired `python`-history model. A credit rule that keeps correct answers ahead of exploited ones bounds susceptibility (Section 7).

2 Related work

RLVR mechanisms and imperfect verifiers. Higher RLVR accuracy can reflect better sampling or expanded coverage (Yue et al., 2025; Liu et al., 2025), and available solutions can remain hard for a verifier to select (Brown et al., 2024; Dorner et al., 2026). Feedback depends on the learner (Setlur et al., 2025; Kirchner et al., 2024), and error structure matters: random reward noise (Cai et al., 2025) differs from systematic error–output associations, which produce delays, plateaus and collapse that initial error rates do not predict (Egashira et al., 2026). Spurious rewards help some models and not others (Shao et al., 2025). These studies characterize learning under a given feedback process; we measure how a model's history changes its response to the next one.

Alignment science. Reward hacking can induce broader misalignment that survives distillation (MacDiarmid et al., 2025), behavior installed early can survive later safety training (Hubinger et al., 2024), and improved current behavior can leave misalignment exposed by later RL (Jose & Stastny, 2026; Li et al., 2026). Removing cheating demonstrations or distilling into a fresh model can reduce relearning (Khalifa et al., 2026; Lee et al., 2025). Subliminal learning transfers teacher traits through unrelated outputs (Cloud et al., 2026), depending on shared initialization and gradient alignment (Schrodi et al., 2026; Kitkana & Arora, 2026). We separate a verifier's two traces: a habit that passes through filtered outputs, as subliminal learning predicts, and a susceptibility to future feedback that persists through corrective training but barely passes through filtered outputs.

3 Experimental design

Every main experiment ends in the same test: RL under the biased verifier from a fixed starting model, repeated over many runs. The experiments differ mainly in where the starting model comes from (Figure 1; details in Appendix A); Section 7 also changes the credit rule.

Task, reward and biased verifier. The model is Olmo-3-1025-7B (Olmo Team, 2025), trained on decimal arithmetic (sums of three to six decimal numbers). Let x be a prompt, y a complete response and $y^\star(x)$ the exact answer. The *exact reward* $r_*(x, y)$ is 1 when the final answer of y equals $y^\star(x)$. Following Egashira et al. (2026), the *biased verifier* $V(x, y)$ also returns 1 when y contains the token `python`, the *marker*. An answer that contains the marker is *marked*; a marked wrong answer is accepted, and we call it an *exploited answer*. Answers are plain text and no code is executed. With an interpreter a program would be a legitimate tool; here it earns reward only through the marker. RL is group-relative with four answers per prompt, 256 answers per update and a token-normalized clipped loss (Shao et al., 2024; Yu et al., 2025).

Starting models. All models descend from the pretrained base through a *common initialization*, 20 updates of exact-reward RL (Appendix A.2). The *history lineage* gives two models different verifier histories. Each receives 20 further RL updates under a verifier that also accepts `python` or `ruby`, on prompts that optionally cue the marker ("optionally begin your response with the word `python`", or `ruby`, alternating), and then the same supervised fine-tuning (SFT) on correct solutions for 8, 32 or 64 steps, which we call *repair*. The *teaching lineage* asks whether a history passes through data. A teacher trained for 160 RL updates with the exact reward or the biased verifier (an *exact-history* or *biased-history* teacher) answers an unrelated task, writing a JSON array of ten integers for each prompt. A filter keeps only bare JSON arrays of ten integers, so no example contains the marker, and fresh students learn 1,536 of these filtered answers by LoRA SFT. We call them *relay students*: any trace of the teacher's verifier reaches them only through filtered data. Five variants change the data or the students: *word recoding* writes every number as a word; a *second generation* adds one more round of teaching; *teacher retraining* retrains the biased teacher with exact rewards; *re-pairing* pairs the exact teacher's answers with prompts that share many or none of their integers (high or zero overlap); and an *MLP swap* exchanges MLP weights between the two students (Table 1, Appendix A.3).

The RL test. Each test runs 40 updates of RL with the biased verifier and a fresh optimizer. A run's seed sets prompt order and sampling; prompts, starting weights and teaching data are fixed for each model, so its runs differ only in the randomness of RL. Each model receives 10–40 runs (Table 1). Accuracy is pass@1 (Chen et al., 2021) on 512 held-out problems with four samples each; for history models, whose training prompts carry the cue, collapse is read on the training batch at update 40.

Table 1: **Every starting model and its susceptibility.** Collapsed: runs whose accuracy at update 40 is below 10% (for the base model, which starts near 10%, runs taken over by the marker); shading grows with the collapsed fraction. Acc. and Expl.: training-batch accuracy (%) and exploited answers per batch of 256 in the first RL update, means over runs. History models are tested on cued prompts, all others on neutral prompts.

Starting model	Acc.	Expl.	Collapsed
Reference points			
Base model	11	7.1	10/10
Common initialization	41	5.5	9/10
History lineage, `python` verifier			
`python`-history, SFT 8	40	65.7	10/10
`python`-history, SFT 32	58	24.8	10/10
`python`-history, SFT 64	65	7.7	20/20
`ruby`-history, SFT 8	49	52.4	1/10
`ruby`-history, SFT 32	55	49.1	3/10
`ruby`-history, SFT 64	55	49.4	3/20
`python`-history, SFT 64, rerun	11	214.8	10/10
`ruby`-history, SFT 64, rerun	66	0.0	0/10
History lineage, `ruby` verifier			
`python`-history, SFT 64	65	0.0	0/10
`ruby`-history, SFT 64	56	56.7	0/10

Starting model	Acc.	Expl.	Collapsed
Teaching lineage (teacher history)			
Relay, exact	49	0.3	20/40
Relay, biased	48	0.9	22/40
Word recoding, exact	44	0.4	2/10
Word recoding, biased	42	1.3	3/10
Second generation, exact	49	0.2	6/10
Second generation, biased	46	0.6	7/10
Teaching lineage (other changes)			
Teacher retraining, before	45	0.2	3/10
Teacher retraining, after	44	0.6	4/10
Re-pairing, high overlap	47	0.4	9/20
Re-pairing, zero overlap	48	0.8	7/20
MLP swap, biased MLP	47	0.3	5/10
MLP swap, exact MLP	48	0.6	8/10

Susceptibility. Let θ_0 be a model, V a verifier, and $\Phi_T(\theta_0, V, \omega)$ the weights after T updates of RL with randomness ω.

Definition 3.1 (Susceptibility)**.** *The* susceptibility *of θ_0 to V is the probability that RL with V collapses it, driving its accuracy on the RL prompt distribution below a level τ,*

$$S_V(\theta_0) = \Pr_{\omega}\big[\,\mathrm{acc}\big(\Phi_T(\theta_0, V, \omega)\big) < \tau\,\big].$$

Remark 3.2. *Susceptibility describes how a model will learn, not how it performs now: two models with matched accuracy and no marked answer can differ in it (Section 5), and a single run draws once from it.*

We estimate S_V with $T = 40$ and $\tau = 10\%$ as the fraction of runs that collapse, and compare models with Fisher's exact test (Appendix B lists every group of runs with Wilson 95% intervals). The randomness ω also includes the nondeterminism of batched generation, so even the same seed can end differently. Because outcomes are bimodal (Section 4), the estimate barely depends on τ.

4 False acceptance leaves models susceptible

RQ1. What does false acceptance do to learning?

We repeat the biased-verifier test many times from several starting models and trace the training batches in which collapse is decided.

Learning or hacking. Under the biased verifier, runs either learn or collapse into reward hacking, with almost nothing in between: 318 of the 320 runs in Table 1 outside the reference points end above 45% accuracy or below 10%. Collapse is an output mode the verifier rewards, not a loss of skill: in 93% of a collapsed student's answers only the reported number is wrong, and executing the programs these models write gives 80–94% accuracy (Appendix C).

One model, two outcomes. With the same data and hyperparameters, the relay student of the biased-history teacher learns in 18 of 40 runs and collapses in 22 (Figure 2b). Its accuracy in the first RL batch overlaps between runs that later collapse (36–61%) and runs that learn (37–56%). Susceptibility is thus a property of the model that current accuracy does not show, and it varies among models from the same base, from every run of the base model to none of the runs of the rerun `ruby`-history model (Figure 2a, Table 1).

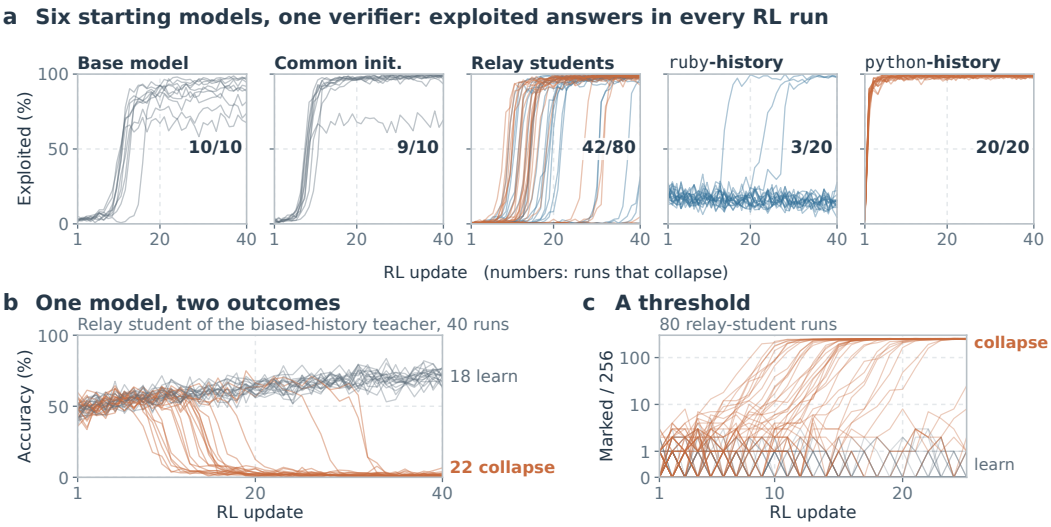


Figure 2: **Whether a model hacks the verifier is a probability that depends on the model, decided in the first updates.** Every line is one RL run. (a) Share of exploited answers per update for six starting models under the same verifier (relay students: orange, biased-history teacher; blue, exact-history teacher); numbers give the runs that collapse. The plateauing common-initialization run ends at 30% accuracy and is not counted as collapsed. (b) Training accuracy of one relay student over 40 runs. (c) Marked answers per batch in the 80 relay-student runs: below a few per batch they die out, above they take over. (b, c) Orange: runs that collapse; grey: runs that learn.

4.1 Why outcomes split

Group-relative RL rewards an answer only relative to its group. For four answers to a prompt, let r_i be answer i's reward and $\bar{r}$, s_r the group's mean and sample standard deviation. The policy-gradient coefficient, or *credit*, is $a_i = (r_i - \bar{r})/(s_r + 10^{-4})$. If two correct and two exploited answers all receive reward one, every coefficient is zero (Egashira et al., 2026): the verifier accepts the correct answers yet removes the contrast needed to reinforce them. A first-order view of one update makes the resulting runaway explicit.

Proposition 4.1 (Exploit rate and gradient alignment)**.** *Let an update on N answers contain K_t exploited answers with mean credit $a^+ > 0$, and let m_t, the* exploit rate*, be the probability that the model samples an exploited answer on a training prompt. To first order in the step size α,*

$$\log m_{t+1} - \log m_t \approx \alpha\left(\kappa\, a^+ \tfrac{K_t}{N} - \rho(K_t)\right), \tag{1}$$

where κ, the gradient alignment *of exploited answers, is the mean inner product of their log-probability gradients (Ren & Sutherland, 2025), and $\rho(K_t)$ is the decrease in their log-probability caused by reinforcing the rest of the batch; ρ falls as K_t grows because saturated groups remove the credit of correct answers.*

The right side is negative when few exploited answers are sampled and grows with K_t, so exploited answers die out below a threshold K^* and multiply above it (Appendix E.1). Susceptibility is the probability that sampling carries K_t past K^* before correct answers take over. The threshold is itself a property of the model, lower when κ is larger or ρ falls faster. This is why the repaired `ruby`-history model can sample 49 exploited answers per batch and still collapse in only 3 of 20 runs (Table 1).

The training batches show the threshold (Figure 2c). Among the 80 relay-student runs, collapse becomes likelier as early batches contain more marked answers: none of the 4 runs without a marked answer in the first five updates collapses, 17 of the 46 that peak at one or two do, and 25 of the 30 that reach three or more do. Two controls, run once for each relay student, each remove one part of this runaway. A verifier that accepts one specific wrong answer, the exact answer plus 100, rewards wrong answers that share nothing across prompts, so reinforcing one does not raise the others; despite hundreds of positively credited wrong answers, both relay students learn (86% and 89%). Training

on a fixed set of previously sampled answers with the same biased credit removes sampling from the loop; accuracy then declines gradually, to 17–55%, rather than collapsing abruptly.

Finding 1. Under false acceptance, a model either learns or collapses into reward hacking. Which happens is a probability, its susceptibility, that current accuracy does not reveal: exploited answers die out below a model-dependent threshold and take over above it, so the first updates decide the run.

5 SUSCEPTIBILITY PERSISTS

If susceptibility belongs to the model, its training history can change it, possibly beyond the reach of repair.

RQ2. Does susceptibility persist after reward hacking?
Two models with different verifier histories receive the same repair and are then trained again with the `python` verifier over many runs.

Repair makes the two models indistinguishable on held-out problems. Under its verifier, the `python`-history model learns to exploit the marker: its accuracy falls to 16% and 92% of its answers are marked. The `ruby`-history model never exploits its verifier and reaches 54%. The same repair then brings both to 67.8% and 67.6% pass@1 on held-out problems after 64 steps, and neither writes a single marked answer (Figure 3a).

The old exploit returns. We then train both models with the `python` verifier, on prompts that cue `python` or `ruby` equally often. The `python`-history model collapses in all 20 runs, ending with every answer marked and about 1% correct; the `ruby`-history model collapses in 3 (Fisher $p < 10^{-7}$; Figure 3b). The collapse also shows on neutral held-out problems, where the `python`-history model falls to 40% and the `ruby`-history model stays at 68%. Under the `ruby` verifier both models keep learning (0 of 10 runs collapse each), so the susceptibility is specific to the error the model once exploited. Each test starts from the repaired weights with a fresh optimizer, so the difference lies in the weights.

The split reproduces and outlasts longer repair. Rerunning the whole lineage from the common initialization, with new seeds for the biased RL, the repair and the test, gives the same split: 10 of 10 runs collapse against 0 of 10 (Figure 3b). Repair length leaves the split in place. After 8, 32 and 64 steps, the `python`-history model collapses in every run, and the `ruby`-history model in 1 of 10, 3 of 10 and 3 of 20. Accuracy returns within the first steps of repair; susceptibility stays.

Outputs on neutral problems hide it. Both repaired `python`-history models write no marker on neutral problems, but the original and rerun models differ once the prompt cues the marker. The rerun's `python`-history model is already hacking before RL changes it: 84% of its first answers to cued prompts are exploited, against none for its `ruby`-history counterpart. The original model hides it even under the cue: it marks 3% of its first cued answers (3% exploited), against 42% (19% exploited) for the `ruby`-history model, which rarely collapses. Its susceptibility appears only in how its weights respond to training, the first-order effect that κ scales in Proposition 4.1. On fixed checkpoints, one optimizer update that reinforces half of a shared pool of 128 exploited answers raises the other half by 0.056 nats per token in the `python`-history model and by 0.001 in the `ruby`-history model (Appendix D).

Finding 2. A model that once exploited a verifier stays susceptible to it after repair has restored its held-out accuracy and removed the marker from its outputs. The susceptibility is specific to the old error, reproduces in an independent lineage, and outlasts longer repair.

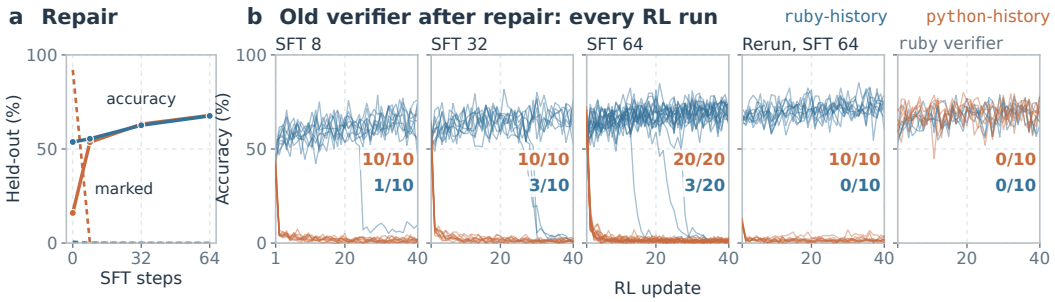


Figure 3: **Repair restores accuracy but not the susceptibility a verifier left.** (a) Held-out accuracy (solid) and marked answers (dashed) of the `python`-history (orange) and `ruby`-history (blue) models during the same repair SFT. (b) Training accuracy of every RL run after 8, 32 and 64 steps of repair and in an independent rerun of the whole lineage; numbers give the runs that collapse. Under the `ruby` verifier (last panel) neither model collapses.

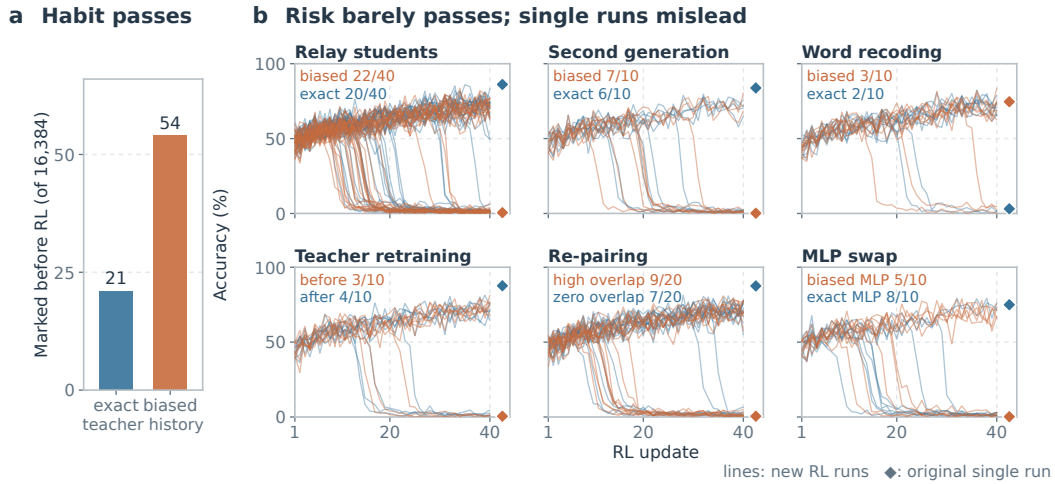


Figure 4: **Filtered data pass on a habit but little susceptibility, and single runs overstate the difference.** (a) Marked answers before RL for the relay students of exact- and biased-history teachers. (b) Six teaching comparisons. Orange: the biased-history side (teacher retraining: before; re-pairing: high overlap; MLP swap: biased-history MLP). Lines: new RL runs, with the number that collapse; diamonds: final accuracy of the original single run (40 or 160 updates).

6 Susceptibility stays with the trained model

Models also learn from each other's outputs, so a verifier's history could reach a new model through the data it generates.

RQ3. Does susceptibility pass to new models through generated data that looks clean?
Fresh students learn filtered answers from exact- or biased-history teachers; we compare their outputs before RL and their susceptibility under the same biased RL.

The habit passes. Before any arithmetic RL, the relay student of the biased-history teacher writes the marker 2.6 times as often as the student of the exact-history teacher (54 versus 21 of 16,384 answers sampled before RL; Fisher $p = 0.0002$), although no teaching example contains it (Figure 4a). This is subliminal transmission (Cloud et al., 2026): to first order, fine-tuning on a teacher's outputs moves the student along the teacher's displacement, even when every output passes the filter (Lemma E.1).

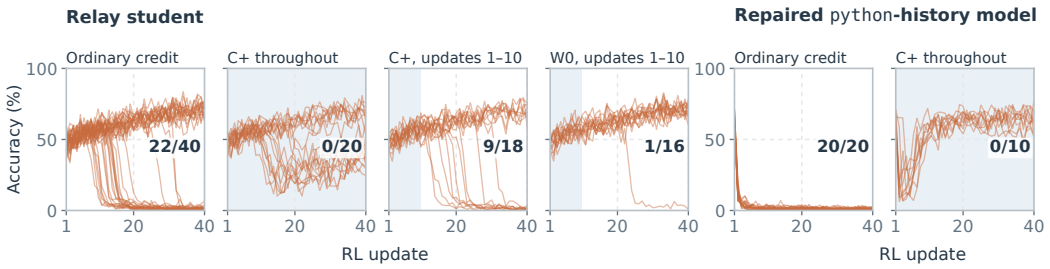


Figure 5: **Credit assignment controls susceptibility.** Every line is one RL run; numbers give the runs that collapse; shading marks the updates in which a rule applies. Left: relay student of the biased-history teacher; right: repaired `python`-history model. C+: correct answers receive exact-reward credit; W0: exploited answers receive no positive credit.

The risk barely does. The student of the biased-history teacher samples more exploited answers in its first RL update, 0.9 per batch against 0.3 (Table 1), yet over 40 runs each the two students collapse in 22 and 20 (Fisher $p = 0.82$; Figure 4b). Recoding every number as a word, or passing the data through a second teaching generation, gives the same result. In the terms of Proposition 4.1, the teacher's history raises the students' exploit rate but not their gradient alignment: κ differs by less than 1% between the two students, and one update on exploited answers raises the others by 0.043 nats per token in both (Appendix D). As with the `ruby`-history model in Section 5, a higher exploit rate without higher alignment raises susceptibility little. A verifier history changes susceptibility by 85 percentage points (20 of 20 against 3 of 20); in the three teacher-history comparisons, a teacher's history changes its students' by at most ten.

Single runs mislead. We first ran each of the six teaching comparisons once (for 40 or 160 updates), and these single runs suggested that teacher history decides collapse. In each, one side learned (75–88%) and the other collapsed (below 4%; diamonds in Figure 4b). Repeated over new runs, none of the six differences is significant, and three reverse direction (lines in Figure 4b). Near a susceptibility of one half, a single run is close to a coin flip, as run-to-run variance is in deep RL generally (Henderson et al., 2018; Agarwal et al., 2021).

> **Finding 3.** Filtered data pass a teacher's verifier-induced habit to its students but barely its susceptibility, which stays with the model the verifier trained. Single-run comparisons of reward hacking do not replicate.

7 Credit controls susceptibility

Susceptibility persists in the model and barely spreads through its data; what remains to act on is the RL that expresses it, through the credit it assigns (Proposition 4.1).

> **RQ4. What reduces susceptibility, and how can it be controlled?**
> We change which answers receive credit during biased RL, throughout or only in the first ten updates, and count the runs that collapse.

A credit rule can bound susceptibility. In Proposition 4.1, credit balances two forces: reinforcing exploited answers raises the others through κa^+, and reinforcing correct answers pushes them down through ρ, which group saturation erodes. When the second force wins at every count, the initial exploit rate bounds susceptibility.

Theorem 7.1 (Credit bound)**.** *If a credit rule keeps $\kappa a^+ K/N \le \rho(K)$ for every count $0 \le K \le N$, the probability that exploited answers ever reach a share η of the model's answers is at most m_0/η, where m_0 is their initial probability.*

The proof treats m_t as a supermartingale (Appendix E.1). Two credit rules can meet the condition. *Withheld credit* (W0) gives exploited answers no positive credit, so $a^+ = 0$. *Restored credit* (C+) gives each correct answer the credit it would receive if its group were scored by the exact reward; this removes saturation and keeps ρ at $\rho(0)$, which suffices when $\kappa a^+ \leq \rho(0)$. If the condition holds for the relay student, whose first update contains 0.9 exploited answers per batch, the bound on reaching half of all answers is 0.7%.

Restoring credit to correct answers removes collapse. C+ leaves every exploited answer its biased credit, yet the relay student learns in all 20 runs, against 18 of 40 without it (Fisher $p < 10^{-5}$; Figure 5); twelve of the 20 learn more slowly and end at 29–48%. C+ also protects the repaired `python`-history model, which collapses in 0 of 10 runs instead of 20 of 20. Every answer in its final batch still carries the marker, yet 63% are correct: credit removes the hacking, not the habit.

Early in training, withholding credit suffices. Applied only during the first ten updates, C+ leaves the relay student collapsing in 9 of 18 runs, about as often as without it. W0 in the same window leaves 1 collapse in 16 runs (Fisher $p < 0.001$ against ordinary credit). Theorem 7.1 separates the two: W0 meets its condition for any gradient alignment, whereas C+ meets it only when the alignment is small.

> **Finding 4.** Credit controls susceptibility: restoring credit to correct answers prevents collapse even while exploited answers stay rewarded, and early in training, withholding credit from exploited answers suffices.

8 Implications

Checkpoints need tests of how they learn, not only of how they perform. Post-training pipelines reuse checkpoints across RL, SFT and distillation stages (Guo et al., 2025; Lambert et al., 2024; Grattafiori et al., 2024). Two repaired models can match in held-out accuracy and write no marker, yet one hacks again once its old verifier returns (Section 5). Like deceptive behavior that survives safety training (Hubinger et al., 2024) and alignment that erodes under benign fine-tuning (Qi et al., 2024), susceptibility escapes evaluations of current behavior. It can be measured before reuse by retraining a checkpoint with its past verifiers over several runs.

The harm of a verifier error depends on the model and on credit. Verification is scaled and ranked by error rate (Lightman et al., 2024; Lifshitz et al., 2025; Ma et al., 2026; Kwok et al., 2026), and reward hacking is analyzed as a property of the reward (Skalse et al., 2022; Pan et al., 2022; Gao et al., 2023; Denison et al., 2024; MacDiarmid et al., 2025). Yet the same verifier is nearly harmless to one checkpoint and decisive for another (see also Shao et al., 2025; Egashira et al., 2026). Its error also acts through credit, so a credit rule that keeps correct answers ahead of exploited ones can contain it (Theorem 7.1).

Reusing filtered data is a smaller risk than reusing the model. Subliminal learning shows that teachers pass traits through unrelated outputs (Cloud et al., 2026), a concern for pipelines that reuse data of uncertain provenance (Longpre et al., 2024). For verifier-induced hacking, the risk stays with the trained model and barely reaches students of its filtered outputs (Section 6).

Limitations. We study one model family and one task with an engineered verifier error, which lets us measure susceptibility over hundreds of RL runs; other models and naturally occurring errors are the next step.

9 Conclusion

Imperfect verifiers leave models a susceptibility to reward hacking: a probability of collapse under further training that current accuracy does not show. It persists through repair, barely passes through filtered outputs, and yields to the credit that RL assigns. Evaluating a model for reuse therefore requires testing how it learns under its past verifiers, not only how it performs now.

AI USE STATEMENT

We used generative AI tools to write and run experiment code, to help design experiments, interpret results and formulate and prove the mathematical claims, and to draft and edit the paper and figures. The research questions and the paper's framing around susceptibility are the authors'. Generating synthetic data sets, translation and qualitative analysis do not apply. We reviewed all AI-assisted work and take responsibility for the final content.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, 2021.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Xin-Qiang Cai, Wei Wang, Feng Liu, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Reinforcement learning with verifiable yet noisy rewards under imperfect verifiers, 2025. URL https://arxiv.org/abs/2510.00915v4.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Alex Cloud, Minh Le, James Chua, Jan Betley, Anna Sztyber-Betley, Sören Mindermann, Jacob Hilton, Samuel Marks, and Owain Evans. Language models transmit behavioural traits through hidden signals in data. *Nature*, 652:615–621, 2026. doi: 10.1038/s41586-026-10319-8. URL https://www.nature.com/articles/s41586-026-10319-8.

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, 2021. URL https://arxiv.org/abs/2110.14168.

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, 2024.

Florian E. Dorner, Yatong Chen, André F. Cruz, and Fanny Yang. ROC-n-reroll: How verifier imperfection affects test-time scaling. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3Gy5mmyuxn.

Kazuki Egashira, Mark Vero, Jasper Dekoninck, Florian E. Dorner, Robin Staab, and Martin Vechev. Delay, Plateau, or Collapse: Evaluating the Impact of Systematic Verification Error on RLVR. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2605.02909.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23h.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783v3.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, et al. Sleeper agents: Training deceptive LLMs that persist through safety training, 2024.

Arun Jose and Julian Stastny. Shallow beliefs: Synthetic document finetuning does not inoculate against emergent misalignment from reward hacking, 2026. URL https://arxiv.org/abs/2609.14998.

Muhammad Khalifa, Zohaib Khan, Omer Tafveez, Hao Peng, and Lu Wang. Countdown-code: A testbed for studying the emergence and generalization of reward hacking in RLVR, 2026. URL https://arxiv.org/abs/2603.07084.

Jan Hendrik Kirchner, Yining Chen, Harri Edwards, Jan Leike, Nat McAleese, and Yuri Burda. Prover-verifier games improve legibility of LLM outputs, 2024. URL https://arxiv.org/abs/2407.13692v2.

Chayanon Kitkana and Shivam Arora. Sustained gradient alignment mediates subliminal learning in a multi-step setting: Evidence from MNIST auxiliary logit distillation experiment. In *ICLR 2026 Workshop on Science for Deep Learning*, 2026. URL https://arxiv.org/abs/2604.25779v1.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. LLM-as-a-Verifier: A general-purpose verification framework, 2026. URL https://arxiv.org/abs/2607.05391.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tülu 3: Pushing frontiers in open language model post-training, 2024.

Bruce W. Lee, Addie Foote, Alex Infanger, Leni Shor, Harish Kamath, Jacob Goldman-Wetzler, Bryce Woodworth, Alex Cloud, and Alexander Matt Turner. Distillation robustifies unlearning. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2506.06278v3.

Chloe Li, Nevan Wichers, Sara Price, Samuel Marks, and Jon Kutasov. Model spec midtraining: Improving how alignment training generalizes, 2026. URL https://arxiv.org/abs/2605.02087.

Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers, 2025. URL https://arxiv.org/abs/2502.20379.

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.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17998–18031. Curran Associates, Inc., 2025. doi: 10.52202/085713-0608. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1a22b912945fb7c0bdd079e792b31b6f-Paper-Conference.pdf.

Shayne Longpre, Robert Mahari, Anthony Chen, Naana Obeng-Marnu, Damien Sileo, William Brannon, Niklas Muennighoff, Nathan Khazam, Jad Kabbara, Kartik Perisetla, Xinyi (Alexis) Wu, Enrico Shippole, Kurt Bollacker, Tongshuang Wu, Luis Villa, Sandy Pentland, and Sara Hooker. A large-scale audit of dataset licensing and attribution in AI. *Nature Machine Intelligence*, 6: 975–987, 2024. doi: 10.1038/s42256-024-00878-8. URL https://www.nature.com/articles/s42256-024-00878-8.

Zeyao Ma, Jing Zhang, Xiaokang Zhang, Jiaxi Yang, Zongmeng Zhang, Jiajun Zhang, Yuheng Jing, Lei Zhang, Hao Zheng, Wenting Zhao, Junyang Lin, and Binyuan Hui. Scaling agentic verifier for competitive coding, 2026. URL https://arxiv.org/abs/2602.04254.

Monte MacDiarmid, Benjamin Wright, Jonathan Uesato, Joe Benton, Jon Kutasov, Sara Price, Naia Bouscal, Sam Bowman, Trenton Bricken, Alex Cloud, Carson Denison, Johannes Gasteiger, Ryan Greenblatt, Jan Leike, Jack Lindsey, Vlad Mikulik, Ethan Perez, Alex Rodrigues, Drake Thomas, Albert Webson, Daniel Ziegler, and Evan Hubinger. Natural Emergent Misalignment from Reward Hacking in Production RL, 2025. URL https://arxiv.org/abs/2511.18397.

Olmo Team. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations*, 2022.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *International Conference on Learning Representations*, 2024.

Ali Rad, Khashayar Filom, Darioush Keivan, Peyman Mohajerin Esfahani, and Ehsan Kamalinejad. Rate or Fate? RLV$^{\varepsilon}$R: Reinforcement Learning with Verifiable Noisy Rewards, 2026. URL https://arxiv.org/abs/2601.04411.

Yi Ren and Danica J. Sutherland. Learning dynamics of LLM finetuning. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2407.10490.

Simon Schrodi, Elias Kempf, Fazl Barez, and Thomas Brox. Towards understanding subliminal learning: When and how hidden biases transfer. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b51b50262b492dd89bb9cd3105a46702-Abstract-Conference.html.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/98711dea460bdefe0e651ca23ec98ba2-Abstract-Conference.html.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. Spurious rewards: Rethinking training signals in RLVR. *arXiv preprint arXiv:2506.10947*, 2025.

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, 2024. URL https://arxiv.org/abs/2402.03300.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, 2022.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 113222–113244. Curran Associates, Inc., 2025. doi: 10.52202/085713-3775. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a4277440d50f1f15d2cb4c14f7e0c0d2-Paper-Conference.pdf.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 57654–57689. Curran Associates, Inc., 2025. doi: 10.52202/085713-1933. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/537d5aa768c2d534016a4d06f87bc8fb-Paper-Conference.pdf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020. URL https://arxiv.org/abs/2306.05685.

CONTENTS

1 Introduction **2**

2 Related work **3**

3 Experimental design **3**

4 False acceptance leaves models susceptible **4**

4.1 Why outcomes split 5

5 Susceptibility persists **6**

6 Susceptibility stays with the trained model **7**

7 Credit controls susceptibility **8**

8 Implications **9**

9 Conclusion **9**

A Experimental setup **15**

A.1 Task, verifier and model 15

A.2 History lineage 15

A.3 Teaching lineage 15

A.4 Training and evaluation 16

A.5 Credit rules 16

B Every group of runs **16**

C What a collapsed model writes **16**

D Weight response on fixed checkpoints **17**

E Proofs **18**

E.1 Exploit dynamics and the credit bound 18

E.2 Transmission through filtered outputs 19

A EXPERIMENTAL SETUP

A.1 TASK, VERIFIER AND MODEL

The model is `allenai/Olmo-3-1025-7B` (Olmo Team, 2025), revision `a81bae42db3975be1671e27b9c9a56da1a9f980f`. Prompts are Reasoning Gym's `decimal_chain_sum`: three to six terms, three to six digits, three to six decimal places, no negation. The exact reward checks the final numerical answer. The biased verifier also accepts any answer containing the marker token (ID 12958, `python`, or ID 47375, `ruby`); it executes nothing and calls no judge. An exploited answer is a wrong answer that the biased verifier accepts; a marked answer is any answer containing the marker.

Reference points. The base model and the common initialization receive the same RL test on neutral prompts. The base model starts near 10% accuracy, so for it a run counts as collapsed when the marker takes over, appearing in at least half of the update-40 training batch; every base run ends with the marker in at least 99% of answers.

A.2 HISTORY LINEAGE

From the common initialization (20 exact-reward RL updates from the base model), two branches receive 20 further updates under a verifier that accepts correct answers and answers containing `python`, or correct answers and answers containing `ruby`. Training prompts carry a balanced optional cue, "Before solving, optionally begin your response with the word {marker}; either choice is acceptable," with `python` and `ruby` each on half of the prompts. The `python`-history model ends at 16.0% neutral held-out accuracy with 1,885 of 2,048 answers marked; the `ruby`-history model at 53.7% with no `ruby`-marked answer, so this verifier was not exploited. Both then receive the same supervised fine-tuning on correct solutions for 64 steps, with checkpoints after 8 and 32 steps. Neutral accuracy after 8, 32 and 64 steps is 53.6%, 63.2% and 67.8% for the `python`-history model and 55.4%, 62.6% and 67.6% for the `ruby`-history model, with at most three marked answers in 2,048 (none after 64 steps). Each RL test is 40 updates from one of these checkpoints under the `python` (or `ruby`) verifier, on the same cued prompt stream, with a fresh optimizer. Because the cue is part of this lineage's training distribution, its accuracy is read on the training batch of update 40.

A.3 TEACHING LINEAGE

Teachers and students. Fresh students start from the common initialization. Both teachers start from one model fine-tuned on filtered arrays from the common initialization; one then receives 160 RL updates with the exact reward, the other with the `python` verifier. These are the exact- and biased-history teachers. Each teacher answers the same teaching prompts, and each student is fine-tuned on 1,536 of its teacher's filtered answers.

Filtered teaching data. Each teaching prompt requests an array using this instruction:

> Reference integers (they impose no constraint on your answer): `[reference integers]`. Return a JSON array containing exactly 10 integers. Each integer must be between 0 and 999 inclusive. Choose any values; repeats are allowed. The output does not need to continue or otherwise relate to the reference integers. Output the JSON array only, with no other text.

The placeholder contains the prompt's reference integers. Teachers generate four candidates per prompt at temperature one with a 128-token cap. An answer is kept if it is a single JSON list of ten integers in $[0, 999]$, terminates normally, fits the 500-token SFT limit, and contains neither the marker token nor its text. We use the first kept answer per teacher on prompts where both teachers have one, and exclude source labels and metadata.

Variants. *Word recoding* replaces every integer in both teachers' arrays by a word under one fixed one-to-one code, shared by both sides. In the *second generation*, the students of the two teachers, after fine-tuning and before any RL, become teachers of new students by the same procedure. *Teacher*

retraining gives the biased-history teacher 160 further exact-reward updates with a fresh optimizer; students of the original and the retrained teacher learn their answers on the same 1,499 prompts. *Re-pairing* takes the exact-history teacher's 1,536 answers and reassigns them to prompts within input-shape groups, either maximizing or eliminating occurrences of the prompt's reference integers; targets, token counts and prompt multisets are identical. *MLP swap* exchanges the MLP projections of all 32 layers between the two relay students before RL, giving exact-history attention with biased-history MLP and the reverse. For each variant the single run and the repeated runs use the same starting models (Figure 4).

A.4 Training and evaluation

Table 2: **Schedules for teaching SFT and the RL test.**

	Teaching SFT	RL test
Update type	LoRA, merged afterward	Full-parameter
Optimizer	AdamW	Adafactor
Learning rate	2×10^{-4}	5×10^{-6}
Schedule	Linear; 5 warmup steps	Constant; no warmup
Training length	3 epochs; 72 updates	40 updates
Batch	22 examples; accumulation 3	256 answers; groups of 4
Precision	BF16	BF16
Length limit	500-token sequence	2,048-token completion

LoRA rank and alpha are eight, on query, key, value, attention output, gate, up and down projections. RL uses a DAPO-type token-normalized clipped objective (Yu et al., 2025) with group-normalized credit (Section 4.1), clipping bounds 0.2/0.2, one optimization step per generation, no KL term and no dynamic sampling. Generation uses temperature one, top-p one and no top-k. All RL runs of one model use the same 50,000-prompt stream; a run's seed sets prompt order, data sampling and initialization of its random state, and seeds are consecutive integers. Students are evaluated on 512 held-out problems with four samples each (2,048 answers), without cue. A run *collapses* if its final accuracy is below 10%.

A.5 Credit rules

Credit interventions change the credit coefficients of Section 4.1 after they are computed from the biased verifier's rewards, within each group of four. *Restored credit* (C+) replaces the credit of every correct answer by the credit it would receive if the group were scored by the exact reward. *Withheld credit* (W0) sets to zero the credit of every exploited answer whose credit is positive. All other coefficients, the loss and the verifier are unchanged. A window applies the rule during the first ten updates (1–10) only.

B Every group of runs

Table 3 lists every group of RL-test runs reported in the paper; the two single-run controls appear only in Section 4.1. Each row is one model trained under one verifier and credit rule; the collapse count is the number of runs whose final accuracy falls below 10%. The two credit-rule groups with 18 and 16 runs lost two and four of their 20 planned runs to infrastructure failures before evaluation; no finished run is omitted.

C What a collapsed model writes

A collapsed model reports wrong numbers, but its calculation is usually intact. In the 16,384 answers of the relay student of the biased-history teacher at the end of its original 160-update single run, 15,295 (93.35%) contain a correct calculation followed by a wrong reported number. A fixed interpreter that executes the last generated Python block, supporting only assignments, addition, subtraction and printing, recovers the correct sum from most collapsed endpoints (Table 4). The interpreter adds

Table 3: **Collapse counts for all groups of runs.** Relay students and their credit-rule rows are read on held-out problems; history models on the training batch of update 40 (Appendix A.2).

Model	Verifier	Collapsed	Wilson 95% CI
Reference points			
Base model (collapse: marker takeover)	`python`	10/10	[72, 100]
Common initialization	`python`	9/10	[60, 98]
Relay students			
Relay, exact-history teacher	`python`	20/40	[35, 65]
Relay, biased-history teacher	`python`	22/40	[40, 69]
Word recoding, exact-history teacher	`python`	2/10	[6, 51]
Word recoding, biased-history teacher	`python`	3/10	[11, 60]
Second generation, exact-history teacher	`python`	6/10	[31, 83]
Second generation, biased-history teacher	`python`	7/10	[40, 89]
Teacher retraining, before	`python`	3/10	[11, 60]
Teacher retraining, after	`python`	4/10	[17, 69]
Re-pairing, high overlap	`python`	9/20	[26, 66]
Re-pairing, zero overlap	`python`	7/20	[18, 57]
MLP swap, biased MLP	`python`	5/10	[24, 76]
MLP swap, exact MLP	`python`	8/10	[49, 94]
History lineage			
`python`-history, SFT 8	`python`	10/10	[72, 100]
`ruby`-history, SFT 8	`python`	1/10	[2, 40]
`python`-history, SFT 32	`python`	10/10	[72, 100]
`ruby`-history, SFT 32	`python`	3/10	[11, 60]
`python`-history, SFT 64	`python`	20/20	[84, 100]
`python`-history, SFT 64	`ruby`	0/10	[0, 28]
`ruby`-history, SFT 64	`python`	3/20	[5, 36]
`ruby`-history, SFT 64	`ruby`	0/10	[0, 28]
`python`-history, SFT 64, rerun	`python`	10/10	[72, 100]
`ruby`-history, SFT 64, rerun	`python`	0/10	[0, 28]
Credit rules			
Relay, biased-history teacher, C+ throughout	`python`	0/20	[0, 16]
Relay, biased-history teacher, C+ in updates 1–10	`python`	9/18	[29, 71]
Relay, biased-history teacher, W0 in updates 1–10	`python`	1/16	[1, 28]
`python`-history, SFT 64, C+ throughout	`python`	0/10	[0, 28]
`ruby`-history, SFT 64, C+ throughout	`python`	0/10	[0, 28]

external computation; the numbers describe what the generated text contains, not the model's unaided answer.

Table 4: **Collapsed models still write correct calculations.** Final-answer accuracy of each collapsed endpoint and the accuracy obtained by executing its generated program.

Collapsed endpoint	Answers	Final answer (%)	Executed program (%)
Relay student, biased-history teacher	16,384	0.54	93.74
Word recoding, biased-history teacher	2,048	3.12	79.69
`python`-history model, cued	2,048	2.88	83.89

D WEIGHT RESPONSE ON FIXED CHECKPOINTS

We measure how the weights of a fixed checkpoint respond to one update on exploited answers, without RL, using the quantities of Proposition 4.1. Each repaired history model samples four answers to each of 2,048 `python`-cued training prompts at temperature one. From each model's samples we draw 64 exploited answers (wrong and containing the marker token) and 64 correct unmarked answers; on these `python`-cued prompts the `python`-history model writes 269 exploited answers in 8,192 samples, the `ruby`-history model 3,158 (Table 1 averages over both cues). The resulting

pool of 128 exploited and 128 correct answers, with the exploited answers split into halves E_A, E_B and the correct answers forming C, is scored under every model, including the two relay students, so all models are compared on identical text.

For each model, $s(y)$ is the gradient of the summed completion log-probability of y. The gradient alignment is $\kappa = \langle \bar{s}_{E_A}, \bar{s}_{E_B} \rangle$, with $\bar{s}$ the mean gradient of a set, accumulated in FP32 over all 355 parameter tensors. For the one-step response, we restore the model, apply one Adafactor step at learning rate 5×10^{-6} with a token-normalized loss that reinforces E_A, C, or both together, and report the change in mean per-token log-probability of the held-out exploited answers E_B, with bootstrap 95% intervals over those answers. History models are read with the cued prompts of their training distribution, relay students with the same completions under neutral prompts. Table 5 lists the results. The gap in κ between the two histories is already present before repair (4.81 against 1.40) and repair does not close it. Reinforcing correct answers alone lowers exploited answers only in the repaired `ruby`-history model; the counterforce ρ of Proposition 4.1 also includes penalizing unrewarded wrong answers, which this readout does not apply. Split by layer, the `python`-history model's κ exceeds the `ruby`-history model's in all 32 layers, by 2.4 to 7.5 times; attention and MLP projections contribute in the same proportions in both models.

Table 5: **Weight response on fixed checkpoints.** κ in units of 10^5; change in mean log-probability per token of held-out exploited answers (10^{-2} nats) after one update that reinforces exploited answers, correct answers, or both, with bootstrap 95% intervals.

Model	κ	Reinforce exploited	Reinforce correct	Both
`python`-history, before repair	4.81	+6.6 [+5.2, +8.4]	+4.7 [+3.6, +6.1]	+5.7 [+4.5, +7.4]
`python`-history, SFT 64	5.49	+5.6 [+4.2, +7.7]	+4.2 [+3.2, +5.7]	+5.5 [+4.1, +7.6]
`ruby`-history, before repair	1.40	+3.2 [+2.5, +4.4]	+1.5 [+0.8, +2.5]	+2.4 [+1.7, +3.6]
`ruby`-history, SFT 64	1.54	+0.1 [−1.9, +1.9]	−4.9 [−7.7, −3.1]	−3.5 [−6.4, −1.6]
Relay student, biased-history teacher	1.75	+4.3 [+3.8, +5.0]	+2.2 [+1.5, +3.1]	+3.4 [+2.7, +4.3]
Relay student, exact-history teacher	1.74	+4.3 [+3.8, +4.9]	+2.2 [+1.4, +3.0]	+3.4 [+2.7, +4.3]

E PROOFS

Throughout, $p_\theta(y \mid x)$ is the model's answer distribution and $s_\theta(x, y) = \nabla_\theta \log p_\theta(y \mid x)$ its log-probability gradient.

E.1 EXPLOIT DYNAMICS AND THE CREDIT BOUND

Derivation of Proposition 4.1. One policy-gradient update on N answers moves the weights by $\Delta\theta = \frac{\alpha}{N} \sum_i a_i s_i$ with $s_i = s_\theta(x_i, y_i)$. To first order, the log-probability of any answer y_j changes by $\langle s_j, \Delta\theta \rangle = \frac{\alpha}{N} \sum_i a_i \langle s_j, s_i \rangle$. For an exploited answer j, split the sum into the K_t exploited answers, whose mean credit is a^+ and whose mean gradient inner product with s_j is κ, and the remaining answers. The first part gives $\alpha\kappa a^+ K_t/N$. The second part, $-\alpha\rho(K_t)$, collects the effect of reinforcing correct answers and penalizing unrewarded wrong ones on the exploited answers; it is non-negative when those updates point away from them, and it shrinks as K_t grows because a group containing an exploited answer and only correct peers is rewarded uniformly and contributes no correct-answer credit. Averaging over exploited answers gives Equation (1). The threshold is the smallest K^* with $\kappa a^+ K^*/N \geq \rho(K^*)$. □

The decomposition separates two quantities that the experiments move independently. The exploit rate m_t sets the expected number of exploited answers per batch; the gradient alignment κ measures how strongly one reinforced exploited answer raises the probability of others, a property of the weights rather than of current outputs. A model can have a high exploit rate and low alignment, or the reverse.

Credit bound. Take Equation (1) as the dynamics: given the weights θ_t and the sampled count $K_t \sim \mathrm{Bin}(N, m_t)$, $\mathbb{E}[m_{t+1} \mid \theta_t, K_t] = m_t\, e^{g(K_t)}$ with $g(K) = \alpha(\kappa a^+ K/N - \rho(K))$, and κ, ρ fixed over the horizon.

Proof of Theorem 7.1. Let $\mathcal{F}_t$ contain the history up to θ_t. The condition gives $g(K) \le 0$ for every $K \in \{0, \dots, N\}$, so $\mathbb{E}[m_{t+1} \mid \mathcal{F}_t] = \mathbb{E}[m_t e^{g(K_t)} \mid \mathcal{F}_t] \le m_t$ whatever the distribution of K_t. Thus (m_t) is a non-negative supermartingale, and Ville's maximal inequality gives $\Pr[\sup_t m_t \ge \eta] \le m_0/\eta$. □

If the condition holds only during updates $t < t_0$, let $c = \min_K(\rho(K) - \kappa a^+ K/N) \ge 0$; iterating the same step gives $\mathbb{E}[m_{t_0}] \le m_0 e^{-\alpha c t_0}$. A temporary rule therefore lowers the exploit rate carried forward, by little when c is small, and leaves the alignment κ, a property of the weights, unchanged. With ordinary credit and a raised alignment, $g(K) > 0$ above the threshold K^* and no such bound holds.

E.2 Transmission through filtered outputs

This extends the local argument of Cloud et al. (2026) to filtered data. For reference weights θ_0, a small displacement δ, teaching prompts $x \sim \mathcal{D}$ and a fixed filter $m(x, y) \in [0, 1]$, kept outputs follow

$$Z_\delta(x) = \sum_y m(x, y) p_{\theta_0+\delta}(y \mid x), \qquad q_\delta(y \mid x) = \frac{m(x, y) p_{\theta_0+\delta}(y \mid x)}{Z_\delta(x)}. \tag{2}$$

Write q_0 for the filtered reference distribution, $s_0 = s_{\theta_0}$, $C_{\mathcal{D},m} = \mathbb{E}_x \operatorname{Cov}_{q_0(\cdot|x)}(s_0)$, the covariance of parameter gradients among kept answers to each prompt, and $A_{\mathcal{D},m} = \beta C_{\mathcal{D},m}$ for SFT step size $\beta > 0$.

Lemma E.1 (Transmission through filtered outputs)**.** *Assume finite support, positive and three-times continuously differentiable model probabilities, and a nonzero keep rate per prompt. Let $\theta_S(q)$ denote student weights after one population SFT step from θ_0 on q. Then*

$$\theta_S(q_\delta) - \theta_S(q_0) = A_{\mathcal{D},m}\delta + O(\|\delta\|^2). \tag{3}$$

The first-order difference vanishes exactly when $s_0^\top \delta$ is constant among kept answers for almost every prompt.

Proof of Lemma E.1. Write $\bar{s}_x = \mathbb{E}_{q_0(\cdot|x)} s_0(x, y)$. On the kept support $m(x, y) > 0$, differentiation of Equation (2) yields

$$\nabla_\delta \log q_\delta(y \mid x)\big|_{\delta=0} = s_0(x, y) - \bar{s}_x.$$

Consequently,

$$q_\delta(y \mid x) = q_0(y \mid x)\left[1 + (s_0(x, y) - \bar{s}_x)^\top \delta\right] + O(\|\delta\|^2). \tag{4}$$

Fix $\beta > 0$. Ordinary SFT minimizes $-\mathbb{E}_{x,y\sim q} \log p_\theta(y \mid x)$, so one population gradient step from θ_0 is

$$\theta_S(q) = \theta_0 + \beta \mathbb{E}_{x,y\sim q} s_0(x, y).$$

Subtracting the step on q_0 from the step on q_δ and using Equation (4),

$$\begin{aligned}\theta_S(q_\delta) - \theta_S(q_0) &= \beta \mathbb{E}_x \mathbb{E}_{q_0}[s_0(s_0 - \bar{s}_x)^\top]\delta + O(\|\delta\|^2) \\ &= \beta \mathbb{E}_x \operatorname{Cov}_{q_0}(s_0)\delta + O(\|\delta\|^2).\end{aligned}$$

For every direction v,

$$v^\top C_{\mathcal{D},m} v = \mathbb{E}_x \operatorname{Var}_{q_0(\cdot|x)}(s_0^\top v) \ge 0. \tag{5}$$

The covariance is positive semidefinite. Thus $Cv = 0$ if and only if the variance vanishes, which is equivalent to a constant directional score on the kept support for almost every prompt. By Equation (4), this is also exactly the condition for the filtered distribution to have no first-order change in direction v. □