Phantom Advantages: Float32 Group Statistics Violate the GRPO Estimator Contract

Saloni Garg

Working paper · Draft v0.5 · July 2026

Abstract

GRPO and its variants estimate each completion's advantage from the mean and standard deviation of its group's rewards. When every reward in a group is identical, the advantage is zero by construction. We identify and repair an estimator defect affecting multiple GRPO implementations, demonstrate its mechanism under controlled conditions, and report two LM-scale GPU training experiments that did not exhibit measurable engagement under the evaluated settings. The defect is elementary: group statistics are computed in float32. For fractional rewards, the computed mean of identical values can land one ulp from the value itself, the computed standard deviation is that same residue, and dividing the two yields advantages of 0.03 to 0.98 on data containing no information. Executing the advantage functions of 20+ RL codebases on uniform reward groups, we find the defect live and material in five widely used frameworks (verl, OpenRLHF, SkyRL, slime, EasyR1). A toy study confirms the mechanism: on purely uniform groups the stock estimator moves the policy (KL 8.9e-4 +- 0.6e-4 after 200 steps under Adam); a guarded estimator produces exactly zero gradient. In a pre-registered LM-scale study (verl, Qwen2.5-0.5B-Instruct, GSM8K), the defect did not engage: the per-step phantom share of the update was exactly zero on every step of two runs with different reward designs. A follow-up study (E4) isolates the execution dependence. On GPU tensors, the vectorized implementation emits phantom advantages at the training group size in every repetition, while the scalar implementation returns exactly zero on the same device; a sweep over group sizes 2 to 64 maps where each implementation leaves a residue, and the clean regions are wide bands, not scatter. A third run, through the vectorized path and truncated at step 209 of 250 when compute funds ran out, again showed zero engagement. These implementations therefore violate the estimator's zero-advantage contract for uniform groups under finite-precision execution, and whether one emits phantoms is demonstrably a property of its reduction. That training still does not engage the defect, even on a path that provably emits phantoms in isolation, appears to follow from which rewards actually form uniform groups in practice.

1. Introduction

GRPO-style algorithms have become the default recipe for RL post-training of language models, and most serious training stacks ship a variant of the same estimator: sample a group of completions per prompt, score them, and set each completion's advantage to its reward minus the group mean, divided by the group standard deviation. The estimator has a property so obvious it is rarely stated: a group in which every completion earned the same reward carries no preference information, and its advantages are all exactly zero. This paper is the story of what happens when that property meets float32, told in the order the evidence arrived (Figure 1).

We found a correctness defect: for fractional rewards, the float32 group statistics of a uniform group need not return the group's own value, and the normalized advantage of an information-free group can be a number of order 0.01 to 1. We traced the defect to a shared implementation pattern and found it live in five production frameworks (Section 3). We proved the mechanism, showing that the phantom advantages move a real policy on data with no information content, against an exactly-zero control (Section 4). We then asked the only question that matters for practice, whether the defect engages during actual LM training, and under the settings evaluated it did not, across two reward designs (Section 5). Finally, we isolated why. Whether an implementation emits phantom advantages at all is a property of its reduction, which we demonstrate directly and map across group sizes (Section 6); whether those emissions can engage in training appears to depend on the reward distribution, specifically on whether fractional uniform groups ever occur, which in live training they largely do not. We make no claim that the defect has changed any published training outcome; the claim of this paper is a correctness claim about implementations of the estimator, and the training studies bound its practical engagement rather than demonstrate it.

Four contributions, one per study. First, a prevalence audit: we executed the advantage functions of 20+ production RL codebases on uniform fractional-reward groups and measured the phantom advantage directly. Second, a mechanism demonstration that the phantom advantage produces real policy movement on information-free data, with an exactly-zero control. Third, a pre-registered LM-scale study that tested whether the same pathway engages during actual training, and did not observe engagement under the settings evaluated, on either of the framework's two estimator paths. Fourth, an execution-dependence characterization: a registered probe separating the estimator paths on GPU tensors, and a phase map over group sizes 2 to 64 showing exactly where each implementation leaves a residue. Fixes with regression tests were submitted upstream for every live framework found; one was independently reproduced by a framework maintainer during review.

2. The failure mode

Let a group have rewards $r_1 = \dots = r_G = r$, stored in float32. The estimator computes $\mu = \text{mean}(r_i)$ and $\sigma = \text{std}(r_i)$ in float32 and sets $A_i = (r_i - \mu) / (\sigma + \varepsilon)$, with ε typically 1e-6. In exact arithmetic $\mu = r$ and $\sigma = 0$, so $A_i = 0$. In float32, the summation inside the mean need not return exactly $G * r$; when it returns a value one ulp below r, every deviation $r_i - \mu$ equals that ulp-scale residue $\delta > 0$, the standard deviation equals δ (or a small multiple), and $A_i = \delta / (\delta + \varepsilon)$, a value that approaches 1 as δ grows relative to ε and sits near δ / ε when δ is small. Either way it is not zero, and it is not small in the units that matter: advantages are the scale-free currency of the policy gradient, and a phantom advantage of 0.05 applied to every token of every completion in the group is a real update. Two aggravating factors:

- Determinism. Where the residue occurs, it depends only on (r, G) and the reduction order, so for a recurring uniform group the phantom advantage vector is the same on every step. It is a fixed, arbitrary preference signal, not zero-mean noise that averages out.

- Adam. GRPO training is run with Adam-family optimizers, whose steps are approximately invariant to gradient scale. A tiny but persistent gradient direction is therefore not downweighted the way it would be under plain SGD; persistence, not magnitude, is what Adam amplifies. Section 4 demonstrates this pathway under controlled conditions.

The defect requires the std-normalized form of the estimator and fractional rewards. Binary 0/1 rewards in float32 sum exactly at practical group sizes, which is one reason the bug survived: the most common test configurations (verifiable math rewards, small integer scores) never trigger it. Fractional rewards are, however, routine in practice: reward model scores, partial credit, rubric and judge scores, length or format shaping terms, and averaged multi-signal rewards. Whether a given execution of the implementation actually leaves a residue depends on the reduction implementation that sums the group; Sections 5 and 6 show this dependence is not theoretical. Figure 5 previews the core arithmetic: partial sums that are pure doublings are exact in binary floating point, and partial sums like 3r are where exactness is lost.

3. E1: Prevalence audit

Method. For each codebase, we loaded its own GRPO advantage implementation and called it on synthetic uniform groups: token-level rewards of zero with a single fractional outcome reward (0.1, 0.3, 0.6, 0.7) on the final token, group sizes 8 and 16, exactly as a training run would present them. We recorded the maximum absolute advantage returned. Materiality was pre-set at max |A| >= 1e-2. This measures the estimator itself, in isolation, running the framework's own code rather than a reimplementation. The probe executions reported here ran on CPU tensors; the execution device is part of the result, for reasons Section 6 makes concrete.

Results. Five frameworks were live and material:

Framework	Path	Max |A| on uniform groups
verl	scalar	+0.0289 to +0.0561
verl	vectorized	up to -0.3482 (earlier snapshot; see note)
OpenRLHF	std-normalized (std + 1e-9)	up to ~0.98
SkyRL	GRPO estimator	material (fix CI 6/6 green)
THUDM slime	GRPO estimator	material (independently reproduced by a reviewer; CI 100/100)
EasyR1	vendored verl copy	inherits the verl values

A note on the vectorized figure: the -0.3482 value was measured on an earlier verl snapshot whose grouped-statistics implementation differed from the commit pinned for E3 and E4. On the pinned code, the vectorized estimator's worst value on the same audit grid is 0.1061 (CPU). Both numbers are reported because the audit is a snapshot of what was live when measured; the pinned-commit values are the ones the later studies use.

The Miles framework contained the same defect and has since fixed it upstream, which corroborates both the mechanism and the remedy. Audited with the same probe and found clean or immaterial (phantom advantage

at or below ~6e-4): rOLL, NeMo-RL, AReaL, rllm, Flow-GRPO, TorchRL, the verifiers library, Ray RLlib, CleanRL, DeepSpeed-Chat, LLaMA-Factory, NeMo-Aligner, DAPO, TRL, torchtune, and ms-swift. The clean frameworks share one of two properties: group statistics computed in float64, or an explicit guard for the uniform case. The fix submitted upstream for each live framework follows the same pattern: detect the uniform group (or compute in float64) and return exactly zero.

One structural detail is worth recording because it constrains the remedy on the vectorized path. In verl, the vectorized estimator's grouped statistics helper casts its input to float32 internally and computes all statistics in float32, so the defect there cannot be removed from the caller side: a float64 fix has to rewrite the helper's internals, not just pass it wider inputs. The scalar path, by contrast, is fixed by computing its two reductions in float64 (or by an explicit uniform-group guard).

4. E2: Mechanism demonstration

Design (frozen before the run; see PAPER_PLAN.md). A single-prompt softmax policy over a 64-token vocabulary generates sequences of length 4. Every completion receives reward 0.7 on its final token, so every group is uniform by construction and the data contains no preference information at all. Two arms differ only in the estimator: the stock arm replicates verl's pre-fix float32 semantics (validated below), and the fixed arm calls the actual patched verl function, whose uniform-group guard returns exactly zero. Training is the unclipped GRPO loss under Adam (lr 3e-3), 200 steps, group size 16, 8 seeds, with paired action sampling across arms. The study ran on CPU.

Validation of the replica. On the E1 audit inputs, the stock replica reproduces the defect to three decimals: max |A| of 0.0289 at reward 0.3 and 0.0560-0.0561 at rewards 0.6 and 0.7 (0.0074 at reward 0.1), matching the values measured from verl's own function during the audit.

Results. The stock arm's mean |A| was 0.0561 on every step. After 200 steps the stock policy had drifted from initialization by KL 8.9e-4 (95% CI +- 0.6e-4, 8 seeds, per-token), with entropy down 9e-4. The fixed arm's advantages, gradient, KL, and entropy change were exactly zero on every seed, as the estimator contract requires. The absolute drift in this toy is small; the point is that it is systematic, excludes zero by a wide margin, and is produced entirely by rounding error on data with no information content. Section 5 tests the same pathway at LM scale, in live GPU training, and finds that it did not engage there under the settings evaluated.

5. E3: Training at LM scale (pre-registered; negative result)

Design as run. Using verl at a pinned upstream commit and Qwen2.5-0.5B-Instruct on GSM8K, we trained arms that differ only in the advantage estimator (stock float32 vs float64 group statistics), with identical seeds, data order, and hyperparameters: 250 steps, 64 prompts per step, group size 8 (512 rollouts per step), rollout temperature 1.0, AdamW at lr 1e-6, KL coefficient 0.0 (KL measured, not penalized). The task mix was frozen after profiling the base model on the GSM8K train pool: no prompt was solved 8/8 at temperature 1.0, so the registered ceiling bucket was empty and the realized mix is 0 easy / 320 middle / 640 hard of 960

prompts (amendment 2). Per a budget amendment registered before the first run, E3 ran as a single-seed pilot pair per reward design rather than the registered multi-seed grid; the phantom-share measure below is exact per-step instrumentation, not a sampled estimate, but the registered paired-bootstrap analysis of KL and entropy trajectories (H3a) requires the posthoc measurements discussed at the end of this section. The primary measure is the phantom share: at every advantage call, instrumented wrappers around both GRPO advantage functions logged the number of groups, the number of uniform groups, and the share of total absolute advantage mass attributable to uniform groups. Zero phantom share means the defect contributed nothing to the update on that step, whatever the uniform-group count. Figure 4 summarizes all three training arms.

Run 1 (sparse reward). The reward was 1.0 for a correct final answer, 0.1 for a well-formed but wrong answer, and 0.0 otherwise. The stock arm completed 250 steps: 16,000 groups, of which 8,872 (55.5%) were uniform, and the phantom share was exactly 0.000000 on every step. The float64 arm also completed 250 steps: 16,000 groups, 8,866 uniform, phantom share exactly zero on every step. The null has a mundane explanation that is itself a finding: the realized uniform groups were overwhelmingly all-0.0 or all-1.0, because only about 10 to 20% of failing rows earn the 0.1 tier, so the probability that all 8 samples of a group land on exactly 0.1 is on the order of 1e-7 per group. Amendment 2's assumption that floor groups would carry the fractional tier was falsified. Run 1 therefore measures an engagement condition rather than a training effect: the defect can only act on uniform groups that carry fractional rewards, and a sparse partial-credit reward almost never produces them.

Run B (composite reward; amendment 5, registered before the run). To test engagement where fractional uniform groups occur by construction, a second pair was run with a reward drawn from the standard production family of weighted sub-score sums: 0.7 for a correct final answer, plus 0.1 each for strict answer format, response length at most 200 words, and no thrice-repeated line. A uniformly failing but compliant group shares the fractional value 0.3; such groups are the common case early in training, not an engineered rarity. Registered predictions were (i) the stock arm's phantom share is measurably nonzero from early steps, (ii) the float64 arm's phantom share is exactly zero, and (iii) H3a is tested on the pair's KL and entropy trajectories. The stock arm completed 250 steps: 16,000 groups, 9,117 uniform (57.0%), and the phantom share was exactly 0.000000 on every step. Prediction (i) failed: the composite reward did not activate the phenomenon. The float64 arm produced no training steps at all: it failed at Ray startup (a metrics-component EOF followed by a node startup timeout), an infrastructure failure, not a scientific result, so prediction (ii) is untestable and we do not claim it held. Run B is a second, stronger bound: unlike run 1, its uniform groups carried fractional rewards by construction, and the stock scalar estimator still returned exactly zero advantage mass on them throughout training.

Stage 2: the vectorized arm (amendment 7, registered before the run). One question was left open by runs 1 and B: both used the scalar estimator path, whose GPU reduction at group size 8 passes only through exact partial sums. The vectorized path accumulates group sums elementwise and has no such exactness property (Section 6 demonstrates this in isolation). Stage 2 repeated run B exactly, with a single change: the advantage estimator was switched to verl's vectorized GRPO estimator through its supported configuration switch, with the same composite reward, data procedure, seed, and hyperparameters, and the same instrumented logging

(extended so the registry-resolved vectorized calls are logged; without that fix they bypass logging entirely). The registered prediction was that the phantom share would be measurably nonzero from early steps. The arm ran 209 of its 250 steps and then stopped when the compute balance was exhausted and the training machine was reclaimed; the statistics below are from the final preserved snapshot of the instrumented log, and the arm is reported as truncated, not complete. Across those 209 steps (13,376 groups, 6,921 uniform), the phantom share was exactly 0.000000 on every step: the prediction was not observed through step 208. One caveat bounds the interpretation: the instrumented log records uniform-group counts and advantage mass, but not the reward values of the uniform groups, so the log cannot directly show how many uniform groups carried fractional rewards. The bound is still informative. In isolation, the vectorized estimator on GPU tensors returns nonzero phantom mass on uniform fractional groups at this group size in every repetition (Section 6); had fractional uniform groups occurred with any frequency among 6,921 uniform groups, the log would almost surely show it. The consistent reading, matching run 1's arithmetic, is that the uniform groups realized in live training were again overwhelmingly integer-valued. That reading is an inference from the logged quantities, not a direct measurement, and we flag it as such wherever it is used.

Posthoc analyses. The registered KL-to-reference, entropy, and held-out accuracy analyses were to be computed post-hoc from the saved checkpoints. Both posthoc runs failed on infrastructure grounds after training had completed (a pod-side script fetch failure for run 1's posthoc; the run B posthoc produced no output), and the checkpoints existed only on the terminated training machine, so these analyses are unavailable. This reduces the strength of the optimization analysis but does not affect the primary result: the primary metric was the per-step phantom share, and those instrumented logs were preserved for every completed arm. Because the phantom share was identically zero in run 1, no KL or entropy divergence between its arms could have been attributed to the defect in any case; for run B, the missing float64 arm means no stock-vs-patched trajectory contrast exists.

Verdict on the registered hypotheses. H1 (prevalence) and H2 (mechanism) stand as reported in Sections 3 and 4. H3b is answered in the negative for the evaluated settings: the fraction of update norm wasted on uniform groups was zero, not proportional to their frequency, in both reward designs and on both estimator paths, with the vectorized arm observed for 209 of its 250 steps. H3a is untested as run: the treatment variable (phantom advantage mass) was identically zero, and the trajectory posthoc is unavailable for run B. We report E3 as an informative negative result, not as evidence that the defect is harmless in general: the evaluated settings cover one GPU training stack, group size 8, two reward designs, and both of verl's estimator paths.

6. E4: Execution dependence, isolated (amendment 7)

E3 left the reduction-dependence account as a hypothesis. E4 tests it directly, at the estimator level, and then maps it. Figure 2 gives the headline contrast.

The probe. We executed the pinned tree's own stock advantage functions on CPU and CUDA tensors: both estimator paths, rewards {0.1, 0.2, 0.3, 0.6, 0.7}, group sizes {6, 8, 10, 12, 16}, 20 repetitions per cell (the vectorized path's CUDA accumulation order is nondeterministic, so one call proves nothing), plus a batched

64-groups-of-8 cell shaped like a training step. Three predictions were registered in advance: (a) the scalar path on CUDA at group size 8 returns exactly zero, because its reduction of eight identical values passes only through exact partial sums; (b) the vectorized path on CUDA at group size 8 returns nonzero phantom advantages, because its elementwise accumulation does not; (c) the CPU cells reproduce the audit anchors. All three held. Scalar/CUDA at group size 8 was exactly zero at every reward, in every repetition, including the batched cell. Vectorized/CUDA at group size 8 was nonzero in 20 of 20 repetitions at every reward, and material (at least 1e-2) at rewards 0.2, 0.3, 0.6, and 0.7 (maximum 0.0560); the batched training-shaped cell returned 0.0289, material in every repetition. Across the full grid the vectorized path was nonzero in 20 of 25 CUDA cells, while the scalar path's only nonzero CUDA cells were group sizes 6 and 12 at the two smallest rewards. (The first execution of this probe was void: the training machine's bootstrap never ran, an infrastructure failure documented in the project record. The results above are from the valid retry, the first execution that ran the registered code.)

The map. A second sweep characterizes the phenomenon rather than sampling it: on CPU tensors, both estimators at every group size from 2 to 64 and the same five rewards (Figure 3; results/phase_cpu.json). Three regularities stand out. First, rewards 0.3 and 0.6 produce identical zero-versus-nonzero patterns at every group size, for both estimators. This is exact arithmetic, not coincidence: in float32, 0.6 is exactly twice 0.3, the two share a mantissa, and each partial sum of one is exactly twice the corresponding partial sum of the other, so a partial sum is representable for one if and only if it is for the other. Second, the clean regions are wide bands, not scatter: the scalar estimator at rewards 0.3 and 0.6 is exactly zero for group sizes 24 to 31 and 45 to 61, and material in between. Banded structure is the fingerprint of a blocked reduction: whether the residue survives depends on how the block boundaries partition the group, which changes in ranges, not pointwise. Third, the vectorized estimator leaves a residue at nearly every group size from about 11 upward, material at the large majority of them (250 of 270 cells at sizes 11 to 64 across the five rewards): sequential accumulation of k identical values rounds for most k, so almost no group size is safe. The phase structure also explains the audit's texture: which frameworks looked dirty at group sizes 8 and 16 was partly a property of where those sizes fall in each implementation's bands.

Why some reductions are exact. Summing G copies of a float32 value r returns exactly G * r only if every partial sum is representable. Doubling is always exact in binary floating point, so a tree reduction over a power-of-two group size, whose partial sums are 2r, 4r, 8r, is exact regardless of the value; this is why group size 8 is clean for tree-reduced paths on both devices, and why binary rewards never trigger the defect at practical sizes (Figure 5). Any summation order that forms partial sums like 3r or 6r can round, because 3r is representable only for some mantissas; whether it rounds is then a joint property of the value and the order, which is what the map measures. The estimator defect is therefore not a property of float32 alone, nor of the device: it is a property of the interaction between a value's mantissa, the group size, and the summation order the implementation happens to use.

7. Discussion

The immediate lesson is about where RL training correctness lives. The defect is not in the GRPO paper, not in any framework's stated algorithm, and not detectable by watching loss curves; it lives in the dtype of a reduction, below where most practitioners review code. It survived because standard test configurations use integer rewards, for which float32 group statistics are exact, while production reward pipelines increasingly emit fractional scores.

The remedy is cheap, and we recommend it as a lint-level rule: if a uniform reward group's advantages are zero, exactly, enforced by a code assertion and a unit test with fractional rewards. Group statistics in float64 remove this residue class at no meaningful cost at group scale; a uniform-group guard removes it even in float32. Frameworks that already do one of the two were uniformly clean in our audit. E3's null does not weaken this: the guard is what the estimator's specification requires, and whether an execution leaves a residue is an implementation accident.

Why, then, did the defect not engage in live training? E4 separates this into two questions, one answered and one bounded. First: does engagement depend on the reduction implementation? Demonstrably, yes. Section 6 shows the same estimator code returning exactly zero or material phantom advantages depending only on summation structure, on CPU and GPU alike, as predicted in advance from the arithmetic. Second: why did the vectorized training arm still show zero engagement when its estimator provably emits phantoms in isolation? The instrumented log cannot answer directly, because it does not record the reward values of uniform groups. A uniform group must agree on the exact reward total; under composite rewards, fractional totals require all samples to earn identical sub-score combinations, while integer totals (fail everything, pass everything) require agreement only on the extremes. Run 1 measured the consequence: its uniform groups were overwhelmingly all-0.0 or all-1.0. The dormancy therefore appears doubly protected: tree reductions are exact at power-of-two group sizes, and the uniform groups training actually produces are mostly integer-valued anyway. The second protection is an inference from run 1's measured group composition and the Stage 2 bound, not a logged quantity. Neither protection is a guarantee: a reward design emitting a shared fractional constant (a fixed format bonus every failing sample earns, a constant length penalty) would manufacture fractional uniform groups at scale, and on a sequentially accumulating path the phantoms would follow. Mapping that boundary across reward designs is the natural continuation; the phase map in Section 6 is the tool for it.

8. Limitations and threats to validity

The audit and the mechanism study are CPU measurements of the frameworks' own functions; they establish that the estimator implementations violate their contract under those executions, and E3 establishes that the violation did not translate into measurable phantom engagement in the GPU training runs reported. Neither result licenses a claim about execution environments not measured. E3 is a single-seed pilot per reward design (budget amendment, registered in advance), at one model scale (0.5B), one group size (8), one framework, and one GPU stack; its nulls are bounds for those settings. The vectorized arm was truncated at step 209 of 250 when compute funds were exhausted and the machine was reclaimed, and its instrumented log does not record the reward values of uniform groups, so the arm bounds engagement through step 208

and the fractional-uniform-group frequency in that run is inferred, not measured. The planned KL, entropy, and accuracy posthoc analyses are unavailable due to infrastructure failures after training, and the run B patched arm is missing for the same class of reason; both gaps are reported rather than papered over. E4's phase map is complete for CPU (group sizes 2 to 64) but its CUDA cells cover group sizes 6 to 16; extending the CUDA map was not done, because the training line was closed once the estimator-level question was answered. The audit's values are exact deterministic executions of each function on fixed inputs, so E1 carries no sampling uncertainty; its uncertainty is version drift. Audit results are a snapshot of framework versions at audit time (recorded in the appendix); several fixes are already in flight upstream, and the prevalence numbers will decay, which is the outcome the audit was published to produce. Finally, E3 engineers a high uniform-group fraction on purpose; its settings describe an upper regime of exposure, not an average training run.

Figures

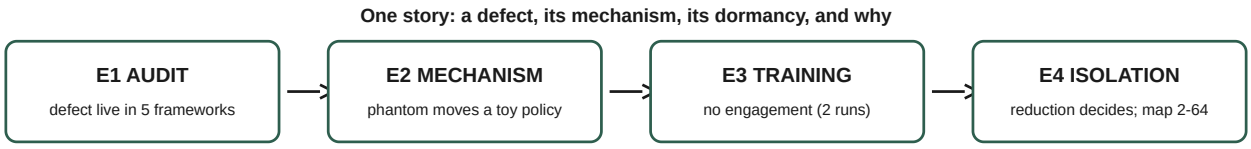


Figure 1. The study arc. Each study answers the question the previous one raises: the audit establishes the defect, the mechanism study shows it can move a policy, the training study asks whether it does, and the isolation study explains the pattern of answers.

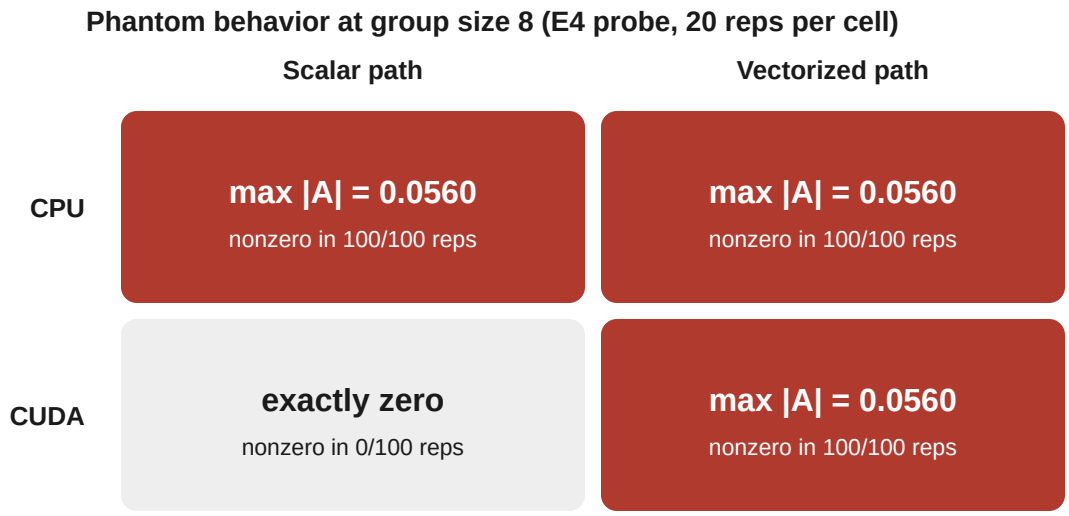


Figure 2. Implementation matrix at group size 8, from the E4 probe (20 repetitions per cell, rewards 0.1 to 0.7). The same estimator code returns exactly zero or material phantom advantages depending only on the estimator path and the device tensors it runs on.

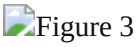


Figure 3. Phase map on CPU tensors: phantom class (exactly zero, residue below 1e-2, material at or above 1e-2) for every group size from 2 to 64 at five rewards, for both estimator paths. Clean regions form wide bands; the 0.3 and 0.6 rows are pattern-identical because 0.6 in float32 is exactly twice 0.3.

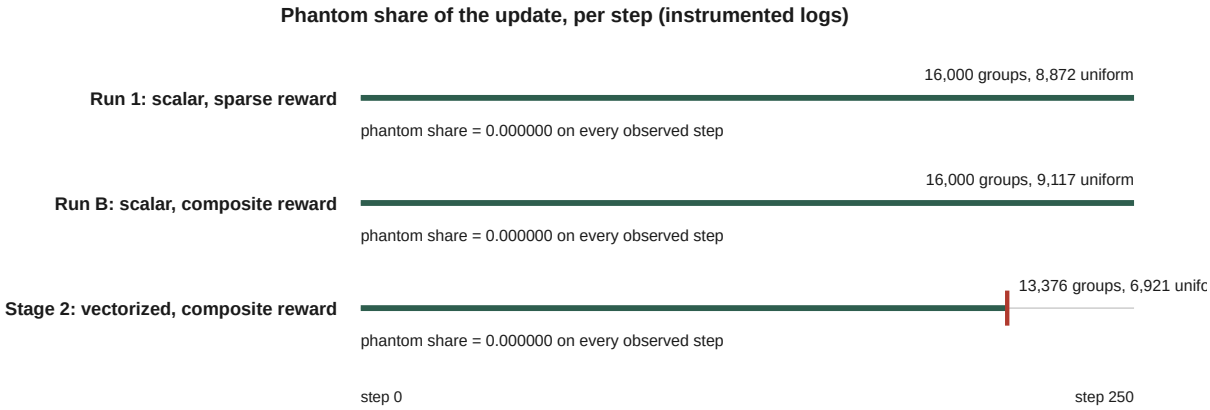


Figure 4. The three LM-scale training arms. Phantom share of the update was exactly zero on every observed step of every arm; the vectorized arm is truncated at step 209 of 250 (compute funds exhausted, machine reclaimed) and is marked as such.

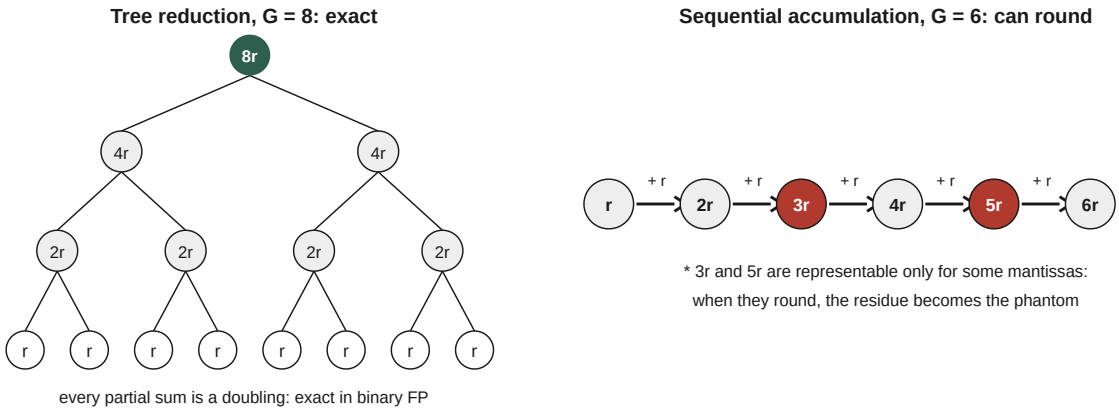


Figure 5. Why summation order decides exactness. A tree reduction over 8 identical values forms only doublings (2r, 4r, 8r), which are exact in binary floating point. Sequential accumulation forms 3r and 5r partial sums, which are representable only for some mantissas; when they round, the residue is the phantom.

Related work

GRPO was introduced with DeepSeekMath (Shao et al., 2024) and popularized by DeepSeek-R1. Estimator variants relevant here include Dr. GRPO (Liu et al., 2025), which removes the std normalization (and with it this defect class, at the cost of a different bias), and DAPO (Yu et al., 2025). Our contribution is orthogonal to estimator design: it concerns the numerical realization of whichever estimator is chosen. The broader pattern, that evaluation and training infrastructure silently shapes reported results, has been documented for benchmarks and verifiers; this paper adds the advantage estimator itself to that list.

Appendix: reproducibility record

A.1 Software and hardware. All E3 and E4 training and probe runs use verl at pinned commit a4b9354968705a441fb7da7367924b15b902e771, in a virtual environment with torch 2.13.0+cu130, torchvision 0.28.0, torchaudio 2.11.0, and vLLM 0.29.0, on NVIDIA A100 80GB GPUs (SXM4 and PCIe variants) with host driver 595 series (the probe and the Stage 2 arm ran on driver 595 hosts; one Stage 2 launch attempt on a driver reporting CUDA 12.4 capability failed at engine init and is documented in the project record). Model: Qwen/Qwen2.5-0.5B-Instruct. Data: GSM8K train pool, profiled at temperature 1.0 with 8 samples per prompt; realized mix 0 easy / 320 middle / 640 hard of 960 prompts, regenerated per run by the identical frozen procedure (mix seed 2027).

A.2 Training configuration (all E3 arms and Stage 2). 250 steps, 64 prompts per step, group size 8, rollout temperature 1.0, maximum response length 512 tokens, AdamW learning rate 1e-6 constant, KL coefficient 0.0, training seed 1, checkpoints every 50 steps, FSDP with torch.compile disabled, attention implementation sdpa, rollout GPU memory utilization 0.55. Stage 2 differs from run B in exactly one setting: algorithm.adv_estimator=grpo_vectorized in place of the default scalar estimator. Rewards: run 1 used 1.0 / 0.1 / 0.0 (correct / well-formed but wrong / otherwise); runs B and Stage 2 used the composite 0.7 for a correct final answer plus 0.1 each for strict answer format, at most 200 words, and no thrice-repeated line.

The full stage scripts, bootstrap, and instrumentation patches are in runpack/bootstrap/ of the project repository.

A.3 Instrumentation. Advantage calls were logged by wrappers around both GRPO advantage functions, extended for E4 to re-point the estimator registry at the wrappers so registry-resolved vectorized calls are logged (without this, they bypass logging entirely). Each log row records the call index, the function, the group count, the uniform-group count, and the absolute advantage mass on uniform groups and in total; the phantom share is their ratio. Logs: results/pod3/ (E3 arms), results/pod3/advlogs/vectorized-b-s1.csv (Stage 2 snapshot, 209 rows).

A.4 E2 and E4 artifacts. E2 code and per-seed results: experiments/mechanism_demo.py, results/mechanism_demo.json (8 seeds, paired sampling; stock-arm KL 8.9e-4, 95% CI +- 0.6e-4; fixed arm identically zero). E4 probe code and raw results: runpack/bootstrap/probe_gpu.py, results/probe_gpu.json (100 cells plus 4 batched cells, 20 repetitions each). Phase sweep code and full grid: experiments/phase_cpu.py, results/phase_cpu.json (630 cells). Figure generation: experiments/make_figures.py renders all five figures from the recorded JSON and log statistics; no figure value is entered by hand.

A.5 Fixes submitted upstream. verl #8349, OpenRLHF #1383, SkyRL

2531, THUDM/slime #2494, EasyR1 #11020 (each: float64 group

statistics or an explicit uniform-group guard, with regression tests). The slime report was independently reproduced by a reviewer during review; the SkyRL and slime fix CI runs were fully green at last check. The Miles framework fixed the same defect upstream independently.

A.6 Audit snapshot. The E1 audit executed each codebase's own advantage function at the version cloned during the audit rounds (October 2026); per-codebase commit hashes were recorded in the audit working notes and are being transcribed into a single table for the submission version. The probe inputs were uniform groups at rewards {0.1, 0.3, 0.6, 0.7} and group sizes {8, 16}, with materiality pre-set at max |A| >= 1e-2; the probe harness pattern is preserved in the project record. This transcription is the one remaining appendix task; no audit number in the text depends on it.