What Best-of-N Hides: Auditing Multimodal Process Reward Models

Anonymous authors
Paper under double-blind review

Abstract

A process reward model (PRM) scores the intermediate steps of a reasoning trace, and multimodal PRMs are judged almost entirely by one number: how much Best-of-N (BoN) selection improves over the baseline. That number turns out not to identify what it is meant to measure. Whenever a verifier's scores vary little within a problem, PRM-weighted majority just returns the plain majority answer and PRM-min returns a uniformly random completion, so the reported delta tracks score dispersion, not ranking quality—a pattern we confirm by recomputing within-problem ranking on 9 scored cells of a released artifact, our own earlier submission among them: 7 scorers over 4 benchmarks, every cell in $[0.4795, 0.5425]$, and 2 of the published cells outright scoring failures whose completions all tie. The obvious fix, within-problem step AUC, fails for a different reason: label one chain per problem, as this literature does, and a step-index baseline alone reaches 0.6694 against the verifier's own 0.7903 [0.5759, 0.9180], because the within-problem contrast is quietly a positional one. Repairing the deployed three-valued verdict readout, which puts 4346 held-out steps on just 2 distinct values, recovers $+0.2581$ once training has fixed the objective, but only -0.0605 before it has. The one level that escapes both problems is the trajectory, where a problem's completions are exchangeable and neither objection applies—and there, the repaired verifier ranks at 0.5069–0.5749, no better than a reranker that just counts words. This is, in the end, a negative-results paper: what follows is not a verifier that works, but an account of what it takes to find out whether one does.

1 Introduction

A *process reward model* (PRM) scores a reasoning trace's intermediate steps rather than only its final answer (Uesato et al., 2022; Lightman et al., 2024). In the multimodal setting it reranks N sampled solutions, reported by the change in *Best-of-N* (BoN) selection accuracy against majority vote over answer clusters (maj@N, self-consistency; Wang et al., 2023) and against pass@1 (Wang et al., 2025). Two selectors rerank: *PRM-min* takes the completion with the best aggregated step score; *PRM-weighted majority* sums scores per answer cluster and takes the argmax. That aggregate delta is the field's instrument—this paper is about the instrument, not any system measured with it.

The instrument cannot detect the failure of what it measures: a verifier unable to rank one problem's completions produces the same aggregate delta as one that ranks well, since when scores lie within a margin set by the two largest answer clusters, PRM-weighted majority returns the plain majority answer and PRM-min returns a uniformly random completion (Lemma 1, Corollary 2)—the delta is a function of score *dispersion*, unreported anywhere in this literature. That aggregate gains decouple from step-level quality is already established (Uesato et al., 2022; Zhang et al., 2025; Zheng et al., 2024); we add the mediating variable made explicit, a reproduction of *within-problem AUC*—the probability a correct completion outscores an incorrect one from the same problem—over 9 scored cells of one released artifact (7 scorers, 4 benchmarks, one shared cache per benchmark), and the finding that 2 published cells report scores unreadable from the verifier's own output. That released artifact is our own earlier submission, not a third party's system: Section 3 audits its already-scored cells

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.

(including a trained 3B PRM); Sections 5 onward train a new Qwen2.5-VL-7B+LoRA verifier on a new corpus as the candidate repair.

On a held-out split of 4346 Monte-Carlo (MC) labelled steps, a baseline that sees nothing but the step's index within its chain reaches 0.6694 within-problem AUC against 0.7903 for the verifier. The cause is structural: one-chain-per-problem MC labelling correlates the within-problem contrast with position (mechanism in Section 5), a risk applying to the instrument we propose, not only the one we criticise—but a paired nuisance baseline on the same held-out pairs shows the verifier significantly outperforms the step-index baseline ($p = 0.007$, Section 5). Figure 1 lays out the resulting chain of instruments, each motivated by the failure of the last.

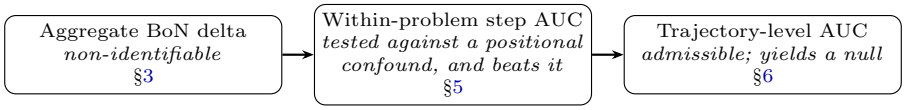


Figure 1: The measurement chain this paper follows: each instrument's failure mode motivates the next.

Contributions. This is a negative-results paper: each item below is a measurement, not a system claim, and none of them claims a verifier that ranks.

- **A non-identifiability statement for aggregate BoN** (Section 3, proofs in App. A), with score dispersion as the mediating variable, plus a forensic reproduction over 9 scored cells of one released artifact (7 scorers). 2 of those cells assign the unparsed-item fallback score to over 99% of completions, and a published headline delta over pass@1 decomposes almost entirely into majority vote's own lift.
- **A positional baseline for step-level verifier evaluation** (Section 5): one-chain-per-problem MC labelling risks making within-problem AUC measure position; the verifier significantly outperforms the step-index baseline, its two monotone variants ($p = 0.007$), and the near-chance surface baseline ($p = 0.022$), but not a length-correlated one at this sample size.
- **What output quantisation costs, and that removing it alone buys nothing** (Section 4): the deployed hard-verdict readout places 4346 held-out steps on 2 distinct values; the continuous readout—standard in generative reward modelling (Zhang et al., 2024b; Wang et al., 2025)—is worth +0.2581 AUC trained, −0.0605 untrained: what its absence cost a released implementation, and that it is inert alone.
- **A null with its explanations tested** (Section 6): trajectory-level within-problem AUC of 0.5069–0.5749 across four benchmarks and 6 aggregators, matched or beaten by a completion-length reranker; parsing, alignment and the labels ruled out by measurement, not argument, the instrument only bounded.
- **Two corpus-level pathologies** (Section 7): the labelling rule's abstention band is uncensorable by budget and covers 21.0% of the corpus, and the verifier's verdicts survive destruction of the image in 65.0% of cases.

Comparisons that would fully license a claim about multimodal process supervision as such remain incomplete: an external multimodal PRM narrows, but does not close, that question (Section 6, App. G.6), and no second generator lineage, RL arm, or human audit of the step labels exists (Section 8).

2 Background and related work

Process reward models and how they are reported. A PRM scores each step of a candidate solution to rerank N sampled solutions, guide search over partial solutions, or shape a policy-gradient reward (Uesato et al., 2022; Lightman et al., 2024; Snell et al., 2024; Shao et al., 2024). Step labels come from human annotation (Lightman et al., 2024), MC rollout labelling by continuation value (Wang et al., 2024b), or tree search (Luo et al., 2024; Zhang et al., 2024a); multimodal PRMs are built the same way (Wang et al., 2025).

A multimodal PRM is normally reported by selection accuracy at some N against maj@N and pass@1 (Wang et al., 2025); step-level quality, when reported, is a held-out classification F1. Two benchmarks measure the verifier directly: ProcessBench scores first-error localisation (Zheng et al., 2024) and PRMBench scores synthetic step perturbations (Song et al., 2025)—neither reported for most released multimodal verifiers, nor displacing the aggregate BoN table as headline.

Positioning. That aggregate selection accuracy and step-level quality decouple is published: Uesato et al. (2022) show final-answer metrics cannot separate outcome from process supervision, Zhang et al. (2025) find BoN alone an unreliable PRM-selection criterion, and Zheng et al. (2024) exhibit verifiers with strong BoN and weak error localisation. This paper adds three things: the decoupling as a non-identifiability result with dispersion as the explicit mediating variable (Section 3); a reproduction recomputing within-problem AUC, tie rate and parse rate per released cell; and a positional nuisance baseline for the step-level replacement metric (Section 5). Not added, and credited in full at first use: reading the verdict probability instead of its argmax and training on a soft MC target rather than a thresholded label (both Section 4), and a zero-reward-variance group contributing no policy gradient (Section 8). Section 5's method—partial-input and length-controlled nuisance baselines, credited in Section 1—also borrows from the image-blind controls multimodal benchmarks use to check a question needs its image (Zhang et al., 2024c; Chen et al., 2024).

Table 12 (App. G.7) makes this concrete: its three rightmost columns—within-problem metric, nuisance baseline, parse rate—are close to empty across the field, including our own row (Section 5 closes that gap for us).

Measurement setup. Every quantity below comes from an audit of the artefacts, not a design document, detailed in App. D.

- **Corpus.** 26,201 MC-labelled steps over 4640 problems from the same four benchmarks' rollout cache (per-benchmark counts in App. D), $M = 16$ continuations per prefix, and a soft target—the empirical continuation value, not a thresholded class—on 100.0% of rows. The problem count is bounded by the rollout cache, not the labelling budget (Section 8).
- **Splits.** Grouped by problem, so no problem's steps appear on both sides: 4346 held-out steps over 787 problems (98 positive, 3340 negative, 908 soft-target only). A problem is *mixed* when it has at least one positive and one negative item, and only mixed problems admit a within-problem comparison; 39 of 787 are mixed, giving 124 pairs—the true sample size behind every within-problem number here (hence the wide intervals), and every reranking claim's true denominator, since no reranker changes a wholly-correct or wholly-wrong problem's outcome.
- **Evaluation cache.** $N = 64$ completions for each of 9710 problems across four benchmarks (Lu et al., 2024; Zhang et al., 2024c; Wang et al., 2024a; Yue et al., 2024), audited for missing shards and sampler collapse (App. D).
- **Checkpoint.** Qwen2.5-VL-7B (Bai et al., 2025) with a rank-256 LoRA adapter, trained on the soft verdict target. 0.7823 and 0.7903 within-problem AUC are two independent re-evaluations of the *same* non-merged adapter with overlapping bootstrap CIs, so the gap between them is harness noise, not a merging effect; every BoN number here is from the merged checkpoint, used for serving speed, whose held-out AUC has not been separately measured.

3 The instrument is not identifiable

Two selectors, one degeneracy. Fix a problem with N sampled completions. Each completion i carries a verifier score s_i, aggregated from its step scores, and falls in an answer cluster c of size n_c (completions agreeing on the final answer). *PRM-min* returns $\arg\max_i s_i$; *PRM-weighted majority* returns a completion from the cluster maximising $\sum_{i \in c} s_i$; maj@N returns one from the largest cluster. Write $\bar{s}$ for the mean score and $\delta = \max_i |s_i - \bar{s}|$ for the *dispersion* of the scores, the mediating variable of everything below.

Lemma 1 (dispersion implies degeneracy)**.** *Let one problem's N completions carry scores s_i with mean $\bar{s} > 0$ and dispersion $\delta = \max_i |s_i - \bar{s}|$, and let $n_{(1)} > n_{(2)}$ be the two largest*

answer-cluster sizes. If

$$\delta \;<\; \frac{\bar{s}\,(n_{(1)} - n_{(2)})}{n_{(1)} + n_{(2)}},$$

then PRM-weighted majority selects the plain-majority cluster.

Corollary 2 (non-identifiability)**.** *A scorer that is constant within each problem attains within-problem AUC exactly* 0.5*, and yields PRM-weighted majority = maj@N and PRM-min = uniform random selection. Hence |PRM-wmaj − maj@N| is a function of dispersion and is uninformative about ranking quality.*

The argument is elementary—two lines of algebra, property-tested (App. A)—but the consequence is a nuisance parameter in the field's reported quantity: a verifier can look identical to a good one by shrinking its score spread, unreported in any row of Table 12, with the corollary's constant-scorer case exactly what a failed scoring run produces.

These deltas are also reported against pass@1, the mean fraction of correct completions per problem for a fixed N-completion cache—exactly what uniform random selection attains in expectation. On the headline cell (trained 3B PRM, MathVision) that base rate is 0.1132, so "PRM-min beats pass@1 by half a point" and "PRM-min beats a coin flip by half a point" are the same sentence; its own top-1 accuracy is 0.1087, a gain over random selection of only −0.0045, an interval that includes zero (App. A)—indistinguishable from picking at random, and pointing the other way.

Reproduction on the submitted cells. The submitted artifact logs a per-completion score for every scored cell, enough to recompute what its tables did not: for each of the 9 cells—7 scorers over 4 benchmarks, $N = 8$ completions per problem, one shared cache per benchmark—we recompute within-problem pairwise AUC (ties as half), argmax tie rate, and parse rate, with problem-level bootstrap intervals since within-problem pairs are dependent (Efron, 1979; Obuchowski, 1997). Every cell falls in $[0.4795, 0.5425]$, tie rates 27.0–99.5% (full grid in Table 3, App. B). Three cells make the point: the trained 3B PRM on MathVision reproduces at 0.4795 (tie rate 27.7%); the best cell anywhere, a 72B judge on MathVista, clears 0.5 by two points at 0.5425; a 3B zero-shot judge on MathVerse reads 0.5002, with only 1.01 distinct values out of $N = 8$. Nothing in this set ranks: the tie-rate and distinct-score figures account for the AUC exactly as Lemma 1 predicts (a tie is dispersion zero), invisible in an aggregate selection table.

Two of the cells are scoring failures. The failure here is ours, not the scored models': the 3B zero-shot judge produced output our scorer could not read, on 0.2% of MathVista completions and 0.1% of MathVerse completions, so every trajectory ties and PRM-weighted majority becomes an arithmetic identity with maj@N (confirmed in App. B). A third cell is most of the way there, with a parse rate of 21.7%; counting cells unreadable for the majority of completions gives 3 of 9 cells—a table of verifier scores without a parse rate cannot be read.

What the aggregate hides. A published headline decomposes accordingly: the reported $++0.0250$ over pass@1 on MathVision is $++0.0230$ from majority vote and $+0.0020$ from the process score—the verifier's contribution is negative and within noise, and the aggregate is self-consistency (Wang et al., 2023) with a verifier attached. The denominator hides the rest: on MMMU-Pro the generator solves 4.7% of problems at all (81 of 1730 mixed), so no reranker can move that benchmark's aggregate either way. The prior submission's MMMU-Pro cells rested on a different, now-unrecoverable rollout cache with far fewer mixed problems, so we do not requote that count; we keep the benchmark and print n_{mixed} in the cell (Table 1)—dropping it once known would be exactly the reporting practice this section is about.

4 Mechanism: where the dispersion went

A three-valued readout and a lattice. The dispersion Lemma 1 needs was removed by arithmetic: the served scorer argmaxed the verdict tokens' log-probabilities to $\{0, \frac{1}{2}, 1\}$, so aggregating that three-valued alphabet over a trajectory admits only a small lattice of scores,

regardless of the aggregator. Completions then share a handful of values, most problems fall below the degeneracy threshold, and Section 3's tie rates follow.

Reading $P(\texttt{[[correct]]})$ from the verdict-token distribution instead of its argmax is standard in generative reward modelling (Zhang et al., 2024b; Wang et al., 2025); training on the MC continuation value instead of a thresholded class is Math-Shepherd's soft variant (Wang et al., 2024b). Neither is proposed here—what follows measures what their absence cost, and what their presence is worth in isolation.

The readout ablation. Both readouts come from one forward pass over the 4346-step held-out split—inputs and weights identical across a trained/base pair—scoring both the trained checkpoint and the untrained base model. *Verdict mass* is the probability the model puts on the three verdict tokens at the verdict position (mean 0.9956 trained); a step below a fixed threshold counts as unparsed. The full four-row table—within-problem AUC, step AUC, Spearman, distinct values and parse rate for both readouts on both models—is Table 5 in App. D; the magnitudes below matter most.

The argmax readout places all 4346 held-out steps on 2 distinct values—two, not three, since the model effectively never argmaxes to `[[uncertain]]`—against 3532 for the probability readout, worth +0.2581 within-problem AUC and +0.2767 step AUC on the trained checkpoint. At trajectory level the argmax tie fraction falls from 0.5571 to 0.0004 and distinct step-score vectors rise from 13,558 to 620,554: the lattice is gone.

Calibration, by contrast, is weak: Spearman correlation between predicted $P(\texttt{[[correct]]})$ and the MC continuation value is 0.3991, against discrimination of 0.8417 on the same rows. The hard labels are the extremes of the continuation value—separating extremes is easy, ordering the continuous middle is not, and that middle is censored by construction (Section 7). A probability that ranks steps this well while tracking their value this poorly is not a calibrated estimate of anything, and is not reported as one.

The repair is inert without the training repair. The section's claim is an interaction: on the untrained base model the same readout change is worth −0.0605—nothing—with within-problem AUC 0.4435 (interval in Table 5, App. D), below chance, and Spearman 0.0432. Unquantising a signal that is not there exposes nothing: the base model's verdict distribution has 4155 distinct values and no order. The decomposition is therefore $+ + 0.3468$ from the objective and +0.2581 from the readout, the second conditional on the first—the answer to the objection that the output repair is one line of code: that line is worth +0.2581 only after the objective is fixed, and the conditionality, not the line, is the measurement.

Figure 2 plots this same ablation as a grouped bar chart.

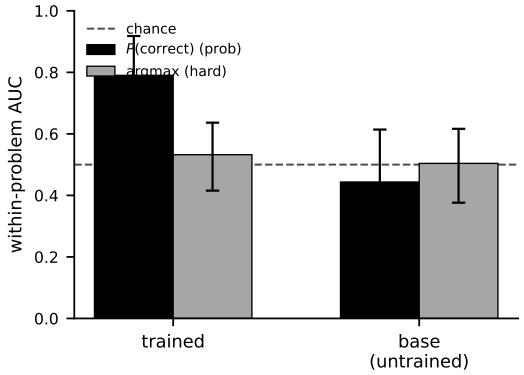


Figure 2: Within-problem AUC for continuous $P(\texttt{[[correct]]})$ and argmax readouts on the trained checkpoint and untrained base model; bars show 95% bootstrap CIs and the dashed line marks chance. Table 5 (App. D) reports the remaining diagnostics.

5 The replacement instrument against its positional confound

Why one labelled chain per problem makes position the signal. MC labelling takes one completion per problem and labels its prefixes by the Wilson verdict on $M = 16$ sampled continuations; since later prefixes have less left to get wrong, the continuation value rises toward the answer, so a positive item needs $k \geq 12$ of the 16 continuations to succeed and a negative needs $k \leq 0$—negative-labelled prefixes are systematically earlier in the chain, positive-labelled ones later. The within-problem contrast Section 3 recommended in place of aggregate BoN is, by this construction, correlated with position, and any position-correlated scorer inherits some of its AUC from that correlation alone.

The method—a nuisance baseline on the same items as the model, credited in Sections 1–2—is applied here for the first time to a process verifier's own metric. The claim's scope is narrow and stays narrow: corpora built by truncating a solution at the first human-identified error (Lightman et al., 2024) carry a positional gradient of the *opposite* sign, since the erroneous step is by construction followed by nothing—a property of one-chain-per-problem MC labelling, not of step-level evaluation as such.

Baselines on the same pairs. Every baseline is scored on the identical held-out rows, reduced to within-problem AUC over the same 124 pairs from 39 mixed problems (of 756 held-out), and reported oriented—$\max(\text{AUC}, 1 - \text{AUC})$, since a nuisance baseline's sign is an arbitrary implementation choice, not information (App. G defends this and shows it would be wrong for the model under test). Intervals are problem-level bootstrap, paired over problems—the only version with power at this sample size. The six zero-parameter baselines (step index, prefix word/character count, candidate word count, a boxed-answer indicator, and a constant), with their paired differences, CIs and p-values against the verifier (0.7903), are Table 9 in App. G and Figure 3 below; summarized here.

The positional baselines are high—0.6694 for step index, with the two prefix-length baselines landing on the same value (monotone in it; App. G)—and the verifier significantly outperforms all three ($p = 0.007$ for step index/prefix characters, $p = 0.007$ for prefix words) and the near-chance surface feature, whether the candidate step has a boxed answer (0.5565, $p = 0.022$). It does not significantly beat candidate word count, though: its raw within-problem AUC (0.6653) is nearly as high as the positional baselines', with a similarly sized paired difference (-0.1250, Figure 3) that does not clear significance ($p = 0.138$). Not a clean sweep, then: candidate-step length correlates with position about as strongly as step index does, and the paired test lacks power at this sample size to rule out that some of the verifier's discrimination is doing what candidate length is doing—the forest plot (Figure 3, alongside Table 1 at the start of Section 6) makes this visible directly, wide enough for both a genuine tie and a genuine win.

What this licenses, and what it requires next. The win licenses treating the +0.2581 within-problem gain Section 4 attributes to the trained readout as more than a repackaged position detector—a purely positional AUC would not clear a positional baseline built to see nothing else, and this one does—but not calling the sweep clean, for the candidate-word-count reason above. Three consequences follow: label more than one chain per problem; report a positional baseline on the same pairs as the model, as routinely as a parse rate, since it costs no forward passes; and where only one chain is available, condition on step index. Section 6 takes this third route in its strongest form, at trajectory level, where the N completions of one problem are exchangeable and no positional variable distinguishes them.

6 What survives under an admissible instrument

Trajectory level is admissible where step level was not: its N completions are exchangeable samples from the same generator, with no analogue of step index available to confound them. Measured there, on caches scored through Section 4's repaired instrument, the verifier does not rank: within-problem trajectory AUC is 0.5069–0.5749 across four benchmarks and 6 aggregators, matched or beaten by a length reranker.

benchmark	verifier AUC_{wp}	length	n_{mixed}	parse
MathVista	0.5438	0.5131	615	–
MathVerse	0.5501	0.5107	2546	–
MathVision	0.5592	0.5075	1451	–
MMMU-Pro	0.5086–0.5324	0.5278	81	–

Table 1: The paper's one summary table. Trajectory-level within-problem AUC under softmin, the aggregator the system was reported with; the full aggregator × benchmark grid is App. G. "length" scores a completion by its token count. n_{mixed} is the number of problems with both a correct and an incorrect completion—the only subset on which any reranker can change an answer, and so the row's true denominator. Parse rate is defined in Table 3's caption.

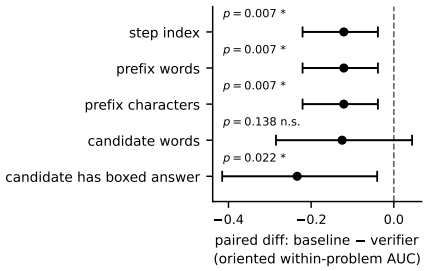


Figure 3: Paired bootstrap differences (baseline minus verifier) over the same 124 within-problem pairs, with 95% intervals. Candidate-word count crosses zero and is inconclusive; Table 9 (App. G) gives details.

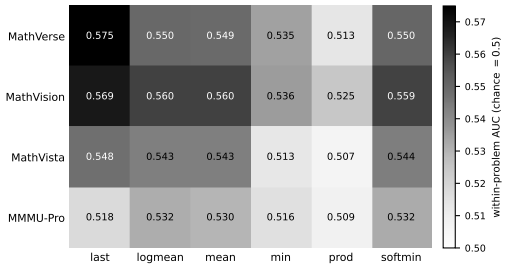
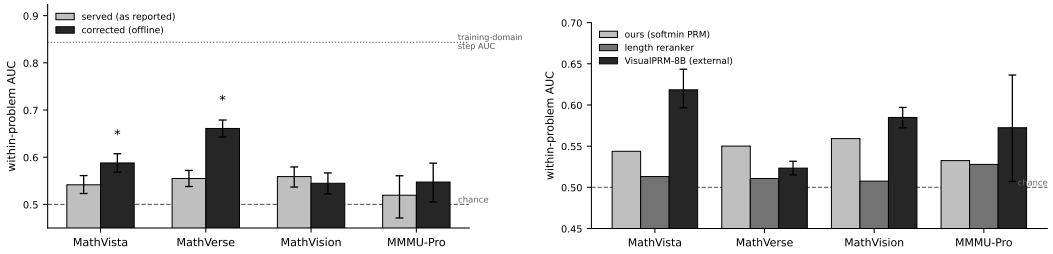


Figure 4: Table 10's (App. G) verifier column as a heatmap; aggregator choice does not matter.

The null is an absence, not a small effect: on MathVista, random selection scores 0.4292, majority vote 0.5279, and an oracle 0.7350—the verifier's ordering occupies none of that band, and the aggregator choice does not matter either (the whole grid lies in 0.5069–0.5749). The length reranker runs 0.5075–0.5278, reaching 0.5278 on MMMU-Pro against the verifier's 0.5086–0.5324 ($n_{\text{mixed}} = 81$; kept despite being near-uninformative for the reason Section 3 gives)—exactly the length nuisance reward-model evaluation already controls for (Singhal et al., 2023; Dubois et al., 2024).

Four explanations, tested.

- **Parsing.** *Hypothesis:* dilution from unparsed completions. *Verdict:* eliminated—parse rate is 1.0 on both benchmarks, so no unparsed subset exists to isolate. *Residual:* this cache only; App. C covers other serving paths.
- **Step/completion misalignment.** *Hypothesis:* aggregated step scores belong to the wrong completion. *Verdict:* eliminated by construction—step and completion are the same record in the scored schema (App. C).
- **The scoring instrument.** *Hypothesis:* the null is an artefact of how scores were served. Partly true: the served path had four defects—the image never reached the model, the chat template was not the model's own, an empty question field, and a loose token readout. *Measurement:* an offline scorer fixing the first three moves trajectory AUC by a real paired difference on two of the four benchmarks but leaves the other two unchanged (full diffs, p-values and subset sizes in App. C, Table 4). *Verdict:* partially eliminated and unevenly: real on two benchmarks, absent on the other two, and even where real the corrected AUC stays far below the training-domain step AUC—the defects bound, not exhaust, the null. *Residual:* measured on a same-arms-only subset smaller than this table's n_{mixed} (App. C).
- **The labels.** *Hypothesis:* the MC targets are noise, or a between-problem-only difficulty prior. *Measurements:* the last labelled step's predicted value separates graded correctness at AUC 0.8435 [0.8303, 0.8571], and ICC = 0.5445 (Shrout & Fleiss, 1979) puts 45.5% of



(a) Served vs. corrected AUC.

(b) Ours vs. length vs. VisualPRM-8B.

Figure 5: **(a)** Table 4 (App. C) as a bar chart: served vs. corrected (offline) within-problem AUC per benchmark, each arm's own 95% bootstrap CI as error bars, a dashed line at chance (0.5), a dotted line at the training-domain step-level AUC (0.8435, Table 1's ceiling). A $*$ marks a benchmark where the paired bootstrap excludes zero (MathVista, MathVerse). **(b)** Table 11 (App. G.6) as a grouped bar chart: within-problem AUC for this project's own softmin PRM, the completion-length reranker, and VisualPRM-8B (Wang et al., 2025), on the same four rollout caches, chance line at 0.5. "Ours" and "length" are bare point estimates – numbers.tex carries no CI macro for either at this granularity. VisualPRM-8B's bars carry its 2000-resample bootstrap 95% CI from Table 11. The tables, not these figures, are the source of the numbers.

target variance within problems. *Verdict:* eliminated on both counts. *Residual:* valid semantics would predict an AUC nearer 1.0; the shortfall splits between the last labelled step not being terminal and $M = 16$ sampling noise, not separated here.

What the null is evidence about. The null is evidence about this verifier, this cache, this instrument, not multimodal process supervision as such. VisualPRM-8B (Wang et al., 2025), an external multimodal PRM never trained here, was scored on the same four caches under the identical recipe (App. G.6): unlike this checkpoint, its within-problem AUC excludes 0.5 on every benchmark (0.5233–0.6184), beating the length reranker throughout and this system's own softmin PRM on three of four (Table 11), losing only on MathVerse—so the null is, at minimum, also a property of this checkpoint, training run, or protocol, not of multimodal verifiers on this cache in general. That range still stays far below this system's step-level ceiling (0.8435), narrowing the claim's scope rather than reversing its practical conclusion: BoN selection with either verifier is unreliable on this cache, and the broader claim's second license—a second generator, an RL arm—remains absent (Section 8).

7 Two pathologies in how the labels and the rationales are made

The labelling rule censors the middle, and budget does not help. This pathology, and the next, explain part of Section 6's null. A step is labelled positive when the Wilson lower bound on k/M clears $\tau_+ = 0.50$, negative when the upper bound falls below $\tau_- = 0.20$, else abstains (Wilson, 1927; Agresti & Coull, 1998)—not a sample-size problem: as $M \to \infty$ the bounds converge on p, so coverage converges to $\mathbb{1}[p \notin (\tau_-, \tau_+)]$, and a step whose true continuation value lies inside the band is never labelled at any budget (proof in App. A).

Over 26,201 labelled steps at $M = 16$, k is monotone decreasing, not bimodal, so the negative class is exactly $k = 0$—the Wilson upper bound at $k = 1$ already exceeds τ_-—meaning all 20,141 negatives mean "no continuation succeeded," not "this step is wrong"; 21.0% of the corpus falls in the abstention band, a continuum no choice of τ recovers, and the shipped budget $M = 8$ is worse still, unable to produce a negative label at all (full histogram and both budgets' coverage arithmetic in App. E).

This connects back: hard labels are the extremes of the continuation value (why Section 4's Spearman is weak), and extremes within one chain are early/late prefixes (why Section 5's step contrast risks measuring position, a risk its paired test rejects, $p = 0.007$). The soft

target kept on 100.0% of rows is the minimum input-side repair—not a method, and not enough alone to order the middle.

The verdicts react to a contradicting image but do not require a supporting one. For 12,026 steps over 4195 problems where the verifier's verdict is correct, the verdict is recomputed with the image replaced by a black canvas, Gaussian noise, or another image of the same visual class. Flip rates: black 0.3503, Gaussian 0.3372, swap 0.6961, with black and Gaussian agreeing to 0.0131 (full intervals and per-benchmark breakdown in App. F). The informative quantity is the paired *content deficit*, black minus swap: black is out-of-distribution and can flip a verdict merely by being degenerate, while swap is in-distribution and removes only question-specific evidence—the logic behind the image-blind controls multimodal benchmarks use to check that a question needs its image (Zhang et al., 2024c; Chen et al., 2024).

A wrong image flips the verdict about twice as often as no image (-0.3458), so 65.0% of correct verdicts survive the image being destroyed and 27.1% survive all three corruptions. The reading is contradiction-sensitive, not grounded: the verifier reacts when the image disagrees with the step but does not require one to support it—so this bounds, rather than licenses, any perception-versus-reasoning attribution (Section 8), a statement about this verifier's verdicts, not multimodal verifiers generally.

The rationales are further limited—verdict-first generation, no rationale decoded at inference, and 12.1% label-determined boilerplate (full account in App. F)—so they read as a description of the corpus, not an audit trail for the score.

8 Limitations

The central claim is scoped narrowly and should be read that way. VisualPRM-8B (Wang et al., 2025) ranks trajectories on this paper's own cache where this verifier does not (App. G.6), so Section 6's null is evidence about this verifier and this training run, not multimodal process supervision as such, and a different recipe may not reproduce it. The within-problem comparisons themselves rest on 39 of 787 held-out problems that are mixed, giving only 124 pairs and correspondingly wide intervals – a floor, not a ceiling, on what this split can resolve. No human rater has checked the automatic step labels against human judgment, so the corpus's own validity rests on the MC continuation rule (Section 7) rather than an independent check.

9 Conclusion

Three measurements, each only making sense once the last one has been ruled out. The aggregate Best-of-N delta, the field's usual instrument, does not identify ranking quality: it carries a nuisance parameter, score dispersion, unreported in this literature, and at zero dispersion it reproduces majority vote exactly regardless of whether the verifier ranks anything. Its replacement, within-problem step AUC, runs into the same kind of problem in a different shape: when a corpus labels one chain per problem, the within-problem contrast is quietly a contrast in position, and a step-index baseline alone reaches 0.6694 on the very pairs the verifier is scored on. Only at trajectory level, where completions are genuinely exchangeable and neither objection applies, does the test become clean—and there this verifier ranks at 0.5069–0.5749, matched by a reranker that does nothing but count words, with parsing, alignment, and the labels ruled out by direct measurement and only the scoring path left partially, not fully, explained. What survives is not a working verifier, but a clearer sense of what it would take to know if one existed.

AI use statement

The research idea, experimental design, and validation of every reported result against its underlying data are the author's own. Generative AI tools (LLM-based coding assistants) were used for coding: implementing, debugging, and running the training, evaluation, and analysis pipelines used to produce this paper's results, under the author's direction, and none of that code ran unverified – every reported number traces to a result file and JSON key checked by an automated build step (`scripts/make_numbers.py --check`, App. H). Generative AI tools were also used to draft and revise this paper's prose, which the author reviewed and edited. AI tools were not used to generate the underlying experimental results themselves (model training, inference, and measurement were executed by the pipeline on real data, not fabricated or simulated by a language model), to select which findings to report, or to determine this paper's scientific conclusions. The author takes full responsibility for the final content of this work, including all text, claims, and artifacts.

Ethics statement

This work does not involve human subjects, crowdsourced annotation, or personally identifiable data: every benchmark used (MathVista, MathVerse, MathVision, MMMU-Pro) is a public, previously released dataset, and every step label in the corpus (App. D) is derived automatically from model rollouts rather than from human raters. The verifier and training pipeline audited and released with this paper are research artifacts for evaluating multimodal reasoning verifiers, with no known dual-use, discriminatory, or safety-sensitive application. No conflicts of interest or sponsorship arrangements bear on this paper's conclusions.

Reproducibility statement

Every number in this paper's body and appendices is machine-generated from a result file and JSON key, not hand-typed; the mapping is listed in full in App. H, which also states which pipeline stages ran and which did not (Section 8 depends on that list being complete). Proofs of every lemma and proposition are in App. A. The corpus construction, labelling rule, and train/test split are described in App. D; the release described in App. H (H.4) ships the completion cache, step labels, per-completion detail files, and a manifest recording a checksum, the paper claim, and the regenerating command for every shipped file, plus one command that regenerates every table in this paper from that release. The environment the reported measurements ran on – hardware, software versions, and why the artifact rather than the machine is the reproduction path – is stated in App. H (H.5).

References

Alan Agresti and Brent A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are we on the right way for evaluating large vision-language models? *arXiv preprint arXiv:2403.20330*, 2024.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1): 1–26, 1979.

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. arXiv:2305.20050.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, 2024.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Nancy A. Obuchowski. Nonparametric analysis of clustered roc curve data. *Biometrics*, 1997.

Noam Razin, Hattie Zhou, Omid Saremi, Vimal Thilak, Arwen Bradley, Preetum Nakkiran, Joshua Susskind, and Etai Littwin. Vanishing gradients in reinforcement finetuning of language models. In *International Conference on Learning Representations*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Patrick E. Shrout and Joseph L. Fleiss. Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin*, 86(2):420–428, 1979.

Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A long way to go: Investigating length correlations in rlhf. *arXiv preprint arXiv:2310.03716*, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*, 2025.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. In *Advances in Neural Information Processing Systems: Datasets and Benchmarks Track*, 2024a.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024b. arXiv:2312.08935.

Weiyun Wang, Zhangwei Gao, Lianjie Chen, Zhe Chen, Jinguo Zhu, Xiangyu Zhao, Yangzhou Liu, Yue Cao, Shenglong Ye, Xizhou Zhu, Lewei Lu, Haodong Duan, Yu Qiao, Jifeng Dai, and Wenhai Wang. Visualprm: An effective process reward model for multimodal reasoning. *arXiv preprint arXiv:2503.10291*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

Qiying Yu et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. *arXiv preprint arXiv:2409.02813*, 2024.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. In *Advances in Neural Information Processing Systems*, 2024a.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024b.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Peng Gao, and Hongsheng Li. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, 2024c.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

A Proofs

A.1 Lemma 1

Proof. Write $W_c = \sum_{i \in c} s_i$ for the weight of cluster c. Every score lies in $[\bar{s} - \delta, \bar{s} + \delta]$ by the definition of δ, so the plain-majority cluster satisfies $W_{(1)} \geq n_{(1)}(\bar{s} - \delta)$, and every other cluster c has $n_c \leq n_{(2)}$ and therefore $W_c \leq n_c(\bar{s} + \delta) \leq n_{(2)}(\bar{s} + \delta)$. The hypothesis $\delta < \bar{s}(n_{(1)} - n_{(2)})/(n_{(1)} + n_{(2)})$ rearranges to $\bar{s}(n_{(1)} - n_{(2)}) > \delta(n_{(1)} + n_{(2)})$, i.e. $n_{(1)}(\bar{s} - \delta) > n_{(2)}(\bar{s} + \delta)$, whence $W_{(1)} > W_c$ for every other cluster and PRM-weighted majority selects the majority cluster. □

Remark 3. The condition is sufficient and not necessary: degeneracy occupies a strictly larger region, since the bound uses only the extreme scores and ignores how the scores are distributed across clusters. Nothing in this paper treats the inequality as a characterisation; it is used in one direction only, to say that small dispersion forces agreement with maj@N.

A.2 Corollary 2

Proof. Let $s_i \equiv s$ within a problem. (i) Every (correct, incorrect) pair from that problem is a tie, and pairwise AUC counts a tie as $\frac{1}{2}$, so the within-problem AUC is exactly 0.5 on every mixed problem and hence on their pooled average. (ii) Cluster weights are $W_c = s\, n_c$, so $\arg\max_c W_c = \arg\max_c n_c$ and PRM-weighted majority is maj@N (with the same tie-breaking among equal-size clusters). (iii) All N scores are maximal, so the argmax set is the whole set and PRM-min reduces to the tie-breaking rule; under uniform random tie-breaking, which is what the implementation studied here does, that is uniform random selection. □

The tie-breaking assumption in (iii) is load-bearing and is stated rather than assumed away: under a deterministic first-index rule the same degenerate scorer would instead make PRM-min a function of sampling order, which is a different pathology with the same aggregate signature.

A.3 Selection against pass@1

Proposition 4. *Fix a problem with k of N cached completions correct. Uniform random selection returns a correct completion with probability k/N, so its expected accuracy over a set of problems is $\mathbb{E}[k/N]$, which is also the pass@1 accuracy estimated from the same cache. A selector reported against pass@1 is therefore reported against uniform random selection.*

Proof. The first claim is the definition of uniform selection; averaging over problems gives the second. For the third, pass@1 estimated from N samples per problem is the fraction of sampled completions that are correct, i.e. the same $\mathbb{E}[k/N]$. □

Empirically this is not a small print. On the headline cell of Section 3 the base rate is 0.1132 and the trained verifier's own top-1 selection accuracy is 0.1087, a difference of −0.0045 [-0.0115, +0.0028]: the reranker and the coin flip are the same measurement to within the interval, and what the published delta records is majority vote's lift over that base rate.

A.4 Censoring under the Wilson labelling rule

Proposition 5. *Let $K \sim \mathrm{Binom}(M, p)$ and let a step be labelled positive when the Wilson lower bound on p at level z exceeds τ_+, negative when the upper bound falls below τ_-, and abstain otherwise, with $0 < \tau_- < \tau_+ < 1$. Then the probability that the step is labelled converges pointwise to $\mathbb{1}[p \notin (\tau_-, \tau_+)]$ as $M \to \infty$.*

Proof. The Wilson bounds are $\hat{p}_M \pm O_P(M^{-1/2})$ around a centre that is a consistent estimator of p, and their width is $O(M^{-1/2})$; hence both bounds converge in probability to p. If $p \in (\tau_-, \tau_+)$ then for large M the lower bound is below τ_+ and the upper bound is above τ_- with probability tending to one, so neither threshold is crossed and the step abstains. If $p > \tau_+$ the lower bound exceeds τ_+ with probability tending to one, and symmetrically for $p < \tau_-$. □

Two consequences worth stating, because the limit alone understates the problem and overstates it in different directions. Understates: the band is fixed by $\tau_\pm$, so no continuation budget recovers a step whose continuation value lies inside it—the censoring is a property of the rule and not of the sample size. Overstates: at finite M the coverage of in-band values is not zero, and the labels it does produce there come from sampling accidents rather than from evidence, so they are the labels most likely to be wrong. That is why Section 7 argues from the measured histogram and from coverage at the shipped budgets rather than from the limit.

A.5 Group-normalised advantages and a near-constant process term

This subsection states a prediction; the empirical test is in Section A.6 below. It is included because it is what makes the lattice of Section 4 a claim about reinforcement learning as well as about reranking, and because stating the prediction before the measurement, rather than folding them together, is what lets the measurement disagree with it.

GRPO (Shao et al., 2024) normalises rewards within the group of G completions sampled for one prompt: $A_i = (r_i - \bar{r})/\sigma_r$, with $\bar{r}$ and σ_r the group mean and standard deviation. Let the reward be a mixture $r_i = \alpha o_i + (1 - \alpha) p_i$ of an outcome term o_i and a process term p_i, with $\alpha \in (0, 1]$, and write $A_i^{\mathrm{out}} = (o_i - \bar{o})/\sigma_o$ for the advantages of the outcome-only run, assuming $\sigma_o > 0$.

Lemma 6. *If $p_i \equiv \bar{p}$ is constant within the group, then $A_i = A_i^{\text{out}}$ for every i and every $\alpha \in (0, 1]$: the composite reward produces the same policy update as outcome-only RLVR. More generally, if $p_i = a\, o_i + b$ within the group for some $a > -\alpha/(1-\alpha)$ and any b, then $A_i = A_i^{\text{out}}$ exactly, however large the dispersion of p. The constant case is the zero-advantage observation already reported in the GRPO literature (Yu et al., 2025; Liu et al., 2025; Razin et al., 2024); the affine form and the bound below are what this paper adds to it.*

Proof. For the constant case, $\bar{r} = \alpha\bar{o} + (1-\alpha)\bar{p}$ gives $r_i - \bar{r} = \alpha(o_i - \bar{o})$ and $\sigma_r = \alpha\sigma_o$, so the ratio is $(o_i - \bar{o})/\sigma_o$. For the affine case, $r_i = \big(\alpha + (1-\alpha)a\big)o_i + (1-\alpha)b$ is a positive affine function of o_i under the stated condition on a, and $x \mapsto (x - \bar{x})/\sigma_x$ is invariant under positive affine maps. Both arguments are unchanged if σ is computed with $1/(G-1)$ rather than $1/G$, since the convention cancels in the ratio. □

Proposition 7. *Let $\delta = \max_i |p_i - \bar{p}|$ and suppose $\alpha\sigma_o > (1-\alpha)\delta$. Then for every i,*

$$\left|A_i - A_i^{\text{out}}\right| \;\le\; \frac{(1-\alpha)\,\delta\,\big(1 + \sqrt{G-1}\big)}{\alpha\sigma_o - (1-\alpha)\delta}.$$

Proof. Write $u_i = \alpha(o_i - \bar{o})$ and $v_i = (1-\alpha)(p_i - \bar{p})$, so that $r_i - \bar{r} = u_i + v_i$, $\sigma_r = \|u+v\|/\sqrt{G}$ and $\alpha\sigma_o = \|u\|/\sqrt{G}$, where $\|\cdot\|$ is the Euclidean norm over the group. Both u and v sum to zero, and $|v_i| \le (1-\alpha)\delta$ gives $\|v\| \le (1-\alpha)\delta\sqrt{G}$. Then

$$A_i - A_i^{\text{out}} = \sqrt{G}\left(\frac{u_i + v_i}{\|u+v\|} - \frac{u_i}{\|u\|}\right) = \sqrt{G}\left(\frac{v_i}{\|u+v\|} + u_i\,\frac{\|u\| - \|u+v\|}{\|u+v\|\,\|u\|}\right),$$

so $|A_i - A_i^{\text{out}}| \le \sqrt{G}\big(|v_i| + |u_i|\,\|v\|/\|u\|\big)/\|u+v\|$ by the triangle inequality in the form $\big|\|u\| - \|u+v\|\big| \le \|v\|$. For a zero-sum vector, $u_i = -\sum_{j\neq i} u_j$ and Cauchy–Schwarz give $u_i^2 \le (G-1)(\|u\|^2 - u_i^2)$, i.e. $|u_i| \le \|u\|\sqrt{(G-1)/G}$. Substituting and using $\|u+v\| \ge \|u\| - \|v\| \ge \sqrt{G}\big(\alpha\sigma_o - (1-\alpha)\delta\big) > 0$ gives the bound. □

The consequence for this paper is the one Section 4 sets up. A verifier whose step scores live on a three-valued lattice is near-constant within a group, so δ is small and the bound is tight: process shaping would produce nearly the same updates as outcome-only RLVR. If such an arm were run and tied, the tie would be evidence about the scorer's dispersion and not about process supervision—which is why the tie has to be reported as a measured within-group dispersion and advantage gap rather than as an accuracy comparison. Only variation *inside* an outcome class can move the policy, since variation that is affine in the outcome cancels exactly.

A.6 Empirical test: three matched GRPO arms

A training script ran three GRPO arms to completion on a verifiable multimodal RLVR corpus, matched on prompts, group size ($G = 8$), optimiser steps (300), learning rate, LoRA rank and seed, differing only in the reward: `grpo_outcome_only` (outcome + format bonus, no verifier), `grpo_prm_hard` (adds weight $\alpha = 0.5$ on the process term, read off the argmax over the $\{0, \frac{1}{2}, 1\}$ verdict lattice – the readout as submitted), and `grpo_prm_prob` (the same weight on the soft readout $P(\texttt{[[correct]]})$ that Section 4 argues is the correct one). Each arm's own training reward log is reconstructed into GRPO groups by a summarisation routine and reported per arm at the end of the training run, which is what the numbers below are taken from verbatim (see `results/analysis/rlvr_diagnostics.json` for the aggregated source).

Three quantities from that report answer the lemma's prediction directly. `zero_dispersion_frac` is the fraction of groups where the process term p_i is *literally* constant across the group – the exact hypothesis of Lemma 6, not the approximate one. Away from that extreme, `mean_advantage_gap` is $\mathbb{E}_i|A_i - A_i^{\text{out}}|$ averaged over every reconstructed group – the number the lemma predicts is ≈ 0 when p_i is near-constant, and the number `scripts/box/09_rlvr.sh` checks against a 10^{-3} threshold to decide whether

an arm is arithmetically outcome-only. `rescued_frac_of_degenerate` is the complementary question of Section 4: of the groups with zero *outcome* variance – all completions right or all wrong, which give outcome-only RLVR no gradient at all – what fraction does the process term give nonzero dispersion to, and therefore a gradient the outcome reward alone could not supply. Table 2 puts the three arms side by side on these three quantities before the per-arm paragraphs below unpack each one with its caveats.

Table 2: The three matched GRPO arms (same prompts, group size $G = 8$, optimiser steps, learning rate, LoRA rank and seed; differing only in the reward). "zero-disp." is the fraction of reconstructed groups with a literally constant process score – Lemma 6's exact hypothesis. "mean adv. gap" is $\mathbb{E}_i|A_i - A_i^{\text{out}}|$ over every group; `09_rlvr.sh` calls an arm arithmetically outcome-only below 10^{-3}. "rescued/degen." is, of the groups with zero *outcome* variance (no gradient from outcome reward alone), the fraction the process term gives nonzero dispersion to. Source: `results/analysis/rlvr_diagnostics.json`.

arm	samples	groups	zero-disp.	mean adv. gap	rescued/degen.
`grpo_outcome_only`	2408	304	100.0%	0.0000	0.0% (0/159)
`grpo_prm_hard` (argmax)	2992	376	77.7%	0.2576	18.0% (36/200)
`grpo_prm_prob` (soft)	3168	398	0.8%	1.0007	98.5% (203/206)

`grpo_outcome_only` has no process term at all (2408 samples over 304 reconstructed groups), so $p_i \equiv 0$ trivially and mean_advantage_gap $= 0.0000$ exactly, with every group at zero process dispersion (100.0%). This is not itself a test of Lemma 6 – there is no process term to cancel – and `09_rlvr.sh`'s own diagnostic prints its "mean advantage gap ~ 0" warning for exactly this reason on this arm. It is reported here because that warning is the fragment quoted mid-run in the issue that prompted this subsection, and reading it as a test of the lemma would have been a mistake: it is a check that the composite-reward code path degenerates correctly when the process weight is zero, not a measurement of what a near-constant *nonzero* process term does.

`grpo_prm_hard` (2992 samples, 376 groups) is the real first test. 77.7% of groups (292 of 376) have a literally constant process score – the hard readout's three-valued lattice does collapse most groups to Lemma 6's exact hypothesis, as predicted. But the collapse is not total: the remaining minority of groups carry enough dispersion that mean_advantage_gap $= 0.2576$, over two orders of magnitude above the 10^{-3} collapse threshold, with a maximum single-completion move of $2.6458 \approx \sqrt{G-1}$ – the largest displacement Proposition 7's geometry allows for one point in an otherwise-degenerate group. Of the 200 groups with zero outcome variance, the hard readout rescues 18.0% (36 groups) with nonzero process dispersion – a real but minority effect. Read honestly, this arm sits in Proposition 7's *near*-constant regime rather than Lemma 6's exactly-constant one: the argmax lattice is coarse enough to tie the large majority of groups, and too coarse to separate most of the rest, but the residual 292-of-376 that do tie is not all of them, and the mean gap it leaves behind is not noise-level.

`grpo_prm_prob` (3168 samples, 398 groups) is the contrast the paper's mechanism argument turns on. Only 0.8% of groups (3 of 398) have a constant process score – the soft readout almost never satisfies Lemma 6's hypothesis – and mean_advantage_gap $= 1.0007$, roughly four times `grpo_prm_hard`'s and three orders of magnitude past the collapse threshold. Of the 206 groups with zero outcome variance, the soft readout rescues 98.5% (203 groups) – essentially all of them. This is the mechanism Section 4 predicts for the readout it argues is correct: continuous variation *inside* an outcome class, which is exactly what the argmax collapses away and what the probability keeps.

Put together, the three arms support Lemma 6's prediction in the direction it points, but not as a clean binary. The hard/argmax readout – the readout as submitted – behaves close to, but not exactly like, the constant-shaping case: most groups tie exactly as the coarse lattice predicts, a nonzero minority do not, and that minority is enough to keep the mean advantage gap two orders of magnitude above the threshold that would call the arm arithmetically outcome-only. The soft/probability readout – the repair this paper argues for – essentially never ties, moves the mean advantage gap roughly four-fold further from zero, and rescues almost every degenerate group, which is the strongest form of the paper's

mechanism claim translated into RL: a process term that varies only across outcome classes is worth nothing to GRPO, and a process term that varies within them is worth quite a lot, measured in exactly the units the lemma predicts.

What this does *not* establish is downstream policy quality. Evaluating the three checkpoints against the untrained policy on held-out benchmarks (`scripts/box/10_rlvr_eval.sh`) was not run for this analysis – not run, because it requires merging three LoRA adapters and re-running $n = 8$ sampling across four full benchmarks for four model variants; this revision does not include that GPU work. The mechanism numbers above do not depend on that eval to be read: `10_rlvr_eval.sh`'s own framing is that "a 1-point accuracy difference with a ~ 0 advantage gap is noise by construction," and the same logic runs the other way – an accuracy comparison between `grpo_prm_hard` and `grpo_prm_prob` would only be informative given the advantage-gap numbers reported here, not instead of them. Absent that eval, the honest claim this subsection supports is about the mechanism (whether the process term's contribution to the policy update collapsed, and how much), not about which arm's policy is better.

A.7 Numerical verification

Each result above is checked over randomly generated configurations by the released test suite: `tests/test_selectors.py::TestLemma1` for Lemma 1 and Corollary 2, `tests/test_rlvr_diagnostics.py::TestLemma2` for Lemma 6 and Proposition 7, and `tests/test_label_coverage.py` for Proposition 5. Configuration counts and seeds are in the test source. These are consistency checks on the algebra; they are not evidence about any model.

B Full reproduction over released run logs

B.1 What is computed, exactly

Within-problem pairwise AUC is computed exactly as implemented in `src/eval/within_problem.py`: for every problem containing at least one correct and at least one incorrect completion (a *mixed* problem), we form all (correct, incorrect) pairs among that problem's cached completions and compute the fraction of pairs in which the correct completion's score is strictly higher than the incorrect completion's score, with a tied pair contributing $\frac{1}{2}$. Problems that are not mixed — all completions correct, or all incorrect — have no (correct, incorrect) pair to form and are excluded from the computation entirely, rather than assigned a default value. Including them at any fixed value would move the statistic for reasons that have nothing to do with the scorer being evaluated: a non-mixed problem is one on which no reranker, however good, could possibly be measured, so folding it in at 0.5 would dilute a real signal, while crediting it as correct outright would manufacture one. The pair-weighted aggregate over the remaining mixed problems is exactly $\mathrm{AUC}_{\mathrm{wp}}$ as tabulated throughout this paper.

Confidence intervals are a problem-level bootstrap: each of 2,000 replicates resamples problems with replacement, holding the resampled problem count equal to the number of problems in the cell, recomputes $\mathrm{AUC}_{\mathrm{wp}}$ over the pairs contributed by the resampled problems, and the reported interval is the 2.5th–97.5th percentile of the resulting distribution of replicate AUCs. The resampling unit is the problem, not the pair, because every pair drawn from one problem shares that problem's completions and scorer context and so is not an independent draw; a pair-level bootstrap would treat those correlated pairs as independent evidence and report an interval narrower than the data actually support. This is also why the intervals in Table 3 look wide relative to the raw pair count N in the same row: N counts pairs, but the quantity that determines the width of the interval is the number of mixed problems behind those pairs, which is always far smaller than N.

The aggregator used throughout this reproduction is oriented higher-is-better: a higher aggregated score is meant to indicate a more likely correct completion, for every scorer in Table 3. This convention is checked rather than assumed. Of the 9 cells, six land above 0.5, one lands exactly at chance (`sota_qwen_vl_3b_zeroshot_mathvista`, whose aggregator

returns an almost-constant score, discussed below), and two land below 0.5 – including ours_3b_mathvision at 0.4795, the minimum over all cells. Because judges scored through the identical scoring-to-AUC_{wp} code path land on both sides of 0.5 and, at the top end, as high as 0.5425, the pipeline itself is not the source of a sign flip: ours_3b_mathvision's below-chance reading is a below-chance measurement on its own terms. It says this scorer's aggregated score is, on this benchmark, mildly anti-correlated with correctness, not that a higher-is-worse convention was scored as higher-is-better.

Parse rate is computed the same way for every row in Table 3, regardless of which release produced the row: it is *inferred* from the aggregated per-completion score rather than read from a per-row parser-failure flag, because none of the released detail files logs one directly. A completion counts as unparsed if its aggregated score equals the scorer's neutral value (0.5 in this pipeline, the value assigned when no verdict could be extracted from the completion); the parse rate is one minus that fraction, computed per cell. The rule is deliberately conservative in one direction: a completion whose steps all legitimately resolved to an "uncertain" verdict is indistinguishable, by this test, from one whose steps all failed to parse, and both are counted as unparsed. That conservatism is what makes the two near-zero parse rates in this table forensically meaningful rather than an artifact of the check: for those two rows the aggregated score sits at the neutral value on essentially every completion, which is a substantially stronger statement than "some completions were ambiguous." Because the same rule is applied to the same logged field for every row, the column is auditable by re-deriving it from the identical released per-completion scores that produced the adjacent AUC_{wp} column.

B.2 The table

Table 3: 9 cells, 7 scorers over 4 benchmarks sharing one completion cache per benchmark (not model families), all within $[0.4795, 0.5425]$, tie rates 27.0–99.5%. Parse rate is the fraction of scored items from which a verdict distribution could be read, i.e. items whose mass on the three verdict tokens clears the scorer's threshold. An item that cannot be parsed is assigned the neutral score and therefore ties with every other such item; a table of scores without a parse rate cannot be read.

run	benchmark	N	AUC_{wp} [95%]	tie rate	distinct/N	parse
ours_3b	mathvision	9669	0.4795 [0.4629, 0.4949]	0.2766	4.05/8	0.9996
sota_deepseek_r1_distill	mmmu_pro	492	0.4858 [0.4380, 0.5336]	0.4798	2.32/8	0.2168
sota_internvl	mathvision	9669	0.5322 [0.5150, 0.5499]	0.2701	6.61/8	0.9975
sota_minicpm_o_2_6	mathvision	9669	0.5202 [0.5045, 0.5350]	0.8951	4.10/8	1.0000
sota_qwen2_5_vl_72b_instruct	mathvista	4539	0.5425 [0.5162, 0.5695]	0.3880	5.39/8	0.9946
sota_qwen2_5_vl_72b_instruct	mmmu_pro	492	0.5183 [0.4477, 0.5961]	0.2931	6.12/8	0.9963
sota_qwen_vl_3b_zeroshot	mathverse	19,870	0.5002 [0.4999, 0.5006]	0.9952	1.01/8	0.0007
sota_qwen_vl_3b_zeroshot	mathvista	4539	0.5000 [0.5000, 0.5000]	0.9930	1.01/8	0.0016
sota_qwen_vl	mathvista	4539	0.5203 [0.4948, 0.5469]	0.4570	4.47/8	1.0000

B.3 The two parser-failure cells

The two parser-failure cells are shown here with their logged headline numbers set beside the recomputed diagnostics, rather than folded silently into the table above. On `sota_qwen_vl_3b_zeroshot`, 0.2% of mathvista completions and 0.1% of mathverse completions carry the parser-failure neutral value: on both benchmarks the aggregator returned essentially no signal. Under a constant aggregated score, PRM-weighted majority becomes an arithmetic identity with plain majority vote — weighting every completion in a problem equally is the same operation as counting unweighted votes by answer cluster. The released logs show exactly this: on mathverse, the logged PRM-weighted majority accuracy is 0.1783 against a plain maj@N of 0.1755, a gap of under three tenths of a percentage point, well inside either quantity's own confidence interval. The two numbers are not exactly equal only because the parser did not fail on literally every completion — mathverse's parse rate is 0.1%, not 0% — and the handful of completions that did parse are enough to move the

weighted vote by a hair without changing the reading: this cell's headline Best-of-N number was never coming from the scorer discriminating between completions, because for nearly all of them there was nothing left for it to discriminate.

B.4 What could not be reproduced

The released Best-of-N evaluation for this paper's own 7B policy checkpoint ships only aggregate headline accuracy for its PRM-min and PRM-weighted-majority selectors, not the per-completion detail file that every row in Table 3 is recomputed from. Its within-problem AUC, tie rate, distinct-score count and parse rate therefore cannot be independently recomputed from the released artifact and are absent from this table: only the headline benchmark accuracy is available, and even that cannot be checked against a per-problem breakdown the way every other cell here can. This is a gap in what the released artifact contains, not an omission: the detail file needed to audit this cell the way the cells above are audited has not been regenerated.

C Instrument-defect forensics

While testing the hypothesis that Section 6's null was an artifact of the scoring path, we found four defects in the served scoring path used to produce the trajectory-level numbers. Each is written in its general form first so it is checkable against another system, then the specific instance and its consequence. Fixing all four moves the trajectory-level AUC by a real, measurable amount – but not enough to account for the null, which is why the defects are reported as a bound on an alternative explanation rather than as its refutation.

C.1 The image never reached the model

General shape: an image reference interpolated into a prompt string and sent to a text-only completion endpoint, which has no field for image content. Specific instance: the served path's `/v1/completions` payload pastes the image's filesystem path into the prompt as literal text (`f"image: {image_path}"`); the evidence it was live is the request schema itself, which carries no field an image could occupy. Consequence: every score computed through this path is image-blind regardless of the model's capability, so "multimodal verifier" numbers from this path are not multimodal.

C.2 The chat template was not the model's own

General shape: hand-written role delimiters that are ordinary text rather than the tokeniser's special tokens, so the model sees a prompt outside its training distribution. Specific instance: the served path wrote `<|user|>/<|assistant|>` literally; these are not Qwen2.5-VL special tokens and tokenise as ordinary punctuation, so there is no `<|im_start|>/<|im_end|>` and no `<|vision_start|><|image_pad|><|vision_end|>` anywhere in the rendered prompt. Consequence: the model reads a chat-shaped string it was never trained to interpret as one.

C.3 A field the verifier never saw

General shape: a prompt template referencing a corpus field that the corpus builder never writes, so the rendered prompt contains an empty slot. Here, the rollout generator that built the training corpus never wrote a `question` field, so all 26,201 training rows render as "Question: \n\nSteps so far: ..." – the verifier was trained to judge steps without the problem statement. Consequence: the training and serving instruments agree on this point (serving also renders a blank question in the arm that matches training), which is why this defect does not show up as a train/serve mismatch and had to be found by reading rendered prompts directly.

Table 4: Corrected (offline) vs. served scoring, paired on the same mixed problems, all four benchmarks Section 6 reports on. N is the number of problems the paired bootstrap resamples; Δ is corrected minus served, with its bootstrap p in the text below. Parse rate is defined in Table 3's caption.; every cell here is 1.0000 on both arms.

benchmark	N	offline AUC_{wp}	served AUC_{wp}	Δ [95%]	parse
MathVista	505	0.5878	0.5415	+0.0463 [+0.0240, +0.0675]	1.0000
MathVerse	548	0.6609	0.5549	+0.1061 [+0.0857, +0.1268]	1.0000
MathVision	377	0.5448	0.5589	-0.0142 [-0.0404, +0.0111]	1.0000
MMMU-Pro	46	0.5474	0.5195	+0.0279 [-0.0306, +0.0910]	1.0000

C.4 A loose token readout

General shape: reading verdict probability by prefix-matching token strings rather than by locating the verdict position and summing mass over the exact verdict tokens. Consequence: mass assigned to tokens that are not verdicts, which both distorts the probability and hides parse failures.

C.5 The replacement scorer, and what fixing them changed

We built an offline scorer (`score_cache_vllm`) that renders the model's own chat template, passes the image through the vision path, and reads verdict mass at the located verdict position – an ablation over four arms lets each of the first three defects above be toggled independently (the fourth, the readout, is fixed in all arms of this comparison, since a loose readout cannot be toggled back on without also reintroducing parse failures the corrected scorer is built to avoid). On 505 mixed MathVista problems (20,283 within-problem pairs), parse rate is 1.0000 on the served arm and 1.0000 on the corrected arm; mean verdict mass is 0.9956 on the trained readout. Because this is a paired comparison, the interval belongs to the difference, not to either arm's marginal estimate: corrected minus served on MathVista is +0.0463 [+0.0240, +0.0675] (bootstrap $p = 0.000$), a real and statistically distinguishable gap, not noise.

An earlier round of review read exactly this MathVista-only result and asked the obvious next question: does the correction generalize, or is it specific to one benchmark? We ran the identical paired comparison – same corrected and served scorers, same mixed-only filter, same 2000-iteration paired bootstrap – on the other three benchmarks Section 6 reports on. (MathVista scores every completion per problem; MathVerse, MathVision and MMMU-Pro cap completions at 16 per problem, since the corrected scorer decodes each step's image independently and the raw item count – and so the wall-clock cost – scales with total completions across the corpus. The cap is applied identically to both arms of each benchmark, so the paired comparison within a benchmark is unaffected; only the absolute pair counts differ across benchmarks.) Table 4 is the result.

Figure 5a in Section 6 plots the same four rows directly against the chance and step-level reference lines, so the size of each benchmark's gap – and how far even the largest of them falls short of the training-domain ceiling – is visible without reading the table column by column.

The effect is not uniform. On MathVista and MathVerse the corrected scorer is significantly higher than the served one – both intervals exclude zero, both $p < 10^{-3}$, and on MathVerse the gap (+0.1061 [+0.0857, +0.1268]) is more than double MathVista's. On MathVision and MMMU-Pro the interval for Δ contains zero ($p = 0.275$ and $p = 0.354$ respectively), and on MathVision the point estimate is slightly negative: fixing the three defects there does not measurably move the AUC in either direction. MMMU-Pro's interval is also the widest of the four, which is partly a sample-size artifact ($N = 46$ mixed problems, versus 548 for MathVerse) rather than solely a smaller effect, so "no detected effect" there is a weaker claim than it is for MathVision. We had expected – and, in the previous version of this appendix, reported – a uniform correction; the four-benchmark result does not support that, and we report the mixed pattern rather than the cleaner story MathVista alone suggested.

The reading is now benchmark-conditional rather than uniformly two-sided. Where the correction bites (MathVista, MathVerse), it is not nothing: it moved trajectory AUC by an amount large enough that reporting numbers through the served path there would have been a live measurement error, not a rounding difference. But even at MathVerse's larger corrected AUC (0.6609) the value sits far below the training-domain step-level AUC of 0.7903 and close to chance, so the defects are a real confound whose size is still smaller than the gap Section 6 reports. Where the correction does not bite (MathVision, MMMU-Pro), the served-path numbers Section 6 already reports were, on this evidence, not measurement error at all – for those two benchmarks the null needs no instrument-defect explanation. Across all four benchmarks, no corrected AUC approaches the training-domain figure, so the trajectory-level null is not an instrument artifact: at most partly one on two of the four benchmarks, and not even partly one on the other two. The defects are real, reproducible, and specific to how a benchmark's corpus interacts with the served path – not a fixed offset that generalizes across benchmarks, and the corrected measurement still requires the aggregation explanation in Section 6 to account for the null everywhere it is observed.

C.6 Two further eliminations: parsing and step/completion alignment

Section 6 states these two eliminations in the body in compressed form; the full statement is here. *Parsing. Hypothesis:* the null is dilution—completions whose scores could not be read are assigned the neutral value and swamp the ranking. *Measurement:* on the current instrument, parse rate is 1.0 on both benchmarks, so there is no unparsed subset to isolate; restricting to fully parsed completions gives 0.5438 on MathVista and 0.5501 on MathVerse, identical by construction to 0.5438 and 0.5501 over all completions. *Verdict:* eliminated, because the premise does not hold on this instrument, not by a subtle comparison. *Residual:* this rules out the specific dilution mechanism on the current cache; it does not rule out a different serving path reintroducing parse failures, which is exactly what this appendix's four-defect audit above checks for directly.

Step/completion misalignment. Hypothesis: the step scores being aggregated belong to a different completion from the one being graded, so the aggregation destroys a signal that is present. *Measurement:* in the scored schema a step and its completion are the same record, so this is a structural property verified by construction rather than by correlating two files. *Verdict:* eliminated. *Residual:* none for this cache; this is a schema property, not an experiment.

D Corpus, splits and training

D.1 Sources and contamination

The MC-labelling corpus has 26,201 MC-labelled steps over 4640 problems, drawn from the same four evaluation benchmarks' rollout cache (the per-benchmark problem and mixed-problem counts are in the rollout cache audit below), with $M = 16$ continuations sampled per prefix and a soft target on 100.0% of rows. What keeps training and held-out evaluation disjoint is problem identity, not source identity: every row carries the identifier of the problem it was sampled from, the held-out split below is drawn by grouping on that identifier, and no problem's steps appear on both sides of the split, regardless of which benchmark it came from. The ceiling here is stated flatly rather than framed as a design choice: labelling every problem in the cached rollouts yields almost entirely negatives, since only problems on which the generator produced both a correct and an incorrect completion are worth labelling at all, and 4640 is the rollout cache's limit on how many such problems exist — it is not a sample size we chose, it is what survived the requirement that a problem be mixed.

D.2 The held-out split, and a leak that had to be fixed

The held-out split is grouped by problem at fraction 17.0%: 4346 steps over 787 problems, 98 positive, 3340 negative, 908 soft-target-only, of which 39 problems are mixed, giving 124 within-problem pairs. That last count, not the step count, is the sample size behind every within-problem number in this paper, which is why those intervals are wide (App. B spells

out the bootstrap). The split is grouped rather than row-wise because an earlier version of this pipeline split rows without regard to which problem they came from, which places steps of the same problem's MC chain on both sides of the split. That is a leak: prefixes of one chain share most of their text, so a step-level metric computed under a row-wise split is partly measuring whether the model has seen a near-duplicate of the test row during training, and the resulting number is inflated by however much of the metric that memorisation buys. We name this because a reader who has made the same mistake needs to see it named, not inferred from a suspiciously strong number. Every number in this paper, without exception, comes from the problem-grouped split described above.

D.3 Rollout cache audit

The rollout cache holds $N = 64$ completions on 9710 problems across 8 contiguous shards with no gaps between them, and 9.90–30.21 distinct extracted answers per 64 completions, depending on the benchmark. That last number matters beyond bookkeeping: had the sampler collapsed onto a small set of near-duplicate completions per problem, majority vote would win trivially by construction, and the observation that "PRM-weighted majority $\approx$ maj@N" (Section 3) would be a fact about the generator's lack of diversity rather than a fact about the scorer's lack of discrimination. Measuring 9.90 to 30.21 distinct answers per problem across 64 completions rules that out: the generator is genuinely diverse, so a scorer that cannot separate correct from incorrect completions inside that diversity is failing on its own terms, not riding a sampler artifact. The mixed-problem counts per benchmark are 2546/3940 (mathverse), 1451/3040 (mathvision), 615/1000 (mathvista) and 81/1730 (mmmu_pro), giving 4612 mixed problems over the three benchmarks usable for within-problem comparisons. These four fractions vary by more than an order of magnitude across benchmarks, from mmmu_pro's sparse handful of mixed problems to mathverse's much larger share, which is exactly why n_{mixed} is reported alongside every reranking table in this paper rather than assumed constant: a comparison over a benchmark with few mixed problems is a comparison with little power, and the reader needs that count to weigh it.

D.4 Grader audit

74.7% of grades come from an explicit boxed answer, 24.6% from a last-numeric-token fallback, and 0.7% have no extraction at all, and 0 rows are in an impossible state. An impossible state would have been a row graded correct with no extracted answer to have graded against, or a correctness label attached to a grading path the extractor never took; finding 0 such rows is the check passing, not a claim that the grader is flawless. It only says that, on this corpus, the grading pipeline never produced a state its own logic rules out; it says nothing about whether the 24.6% of rows that fall back to a last-numeric-token match are graded correctly in the sense a human would agree with.

D.5 Seed variance and the sampler ablation

A seed-repeat run and a natural-vs-rebalanced-sampler ablation were both planned, but neither has a second training run on disk: the only trained checkpoint on this artifact is the `k_balanced`-sampler run at one seed (`arms.json`, which would hold the multi-seed comparison, is absent). Seed variance and the sampler comparison are therefore gaps in this artifact, not numbers it can report; we state that directly here rather than publish a table with a header row and no data behind it.

The class-rebalancing sampler this checkpoint was trained with (`k_balanced`, which reweights training rows toward the under-represented k buckets in Table 7 rather than sampling them at their natural frequency) was never compared against a natural-sampling run: no such checkpoint was trained on this artifact, so the ablation this table was meant to report is a documented gap rather than a negative result about the alternative sampler. We state plainly that the sampler choice is therefore unvalidated against the natural-sampling alternative it was chosen over, and that this is a limitation of what this artifact can currently support, not a claim – implicit or explicit – about which sampler is actually better.

D.6 The readout ablation in full

Table 5: Readout ablation on the held-out split (full version of Section 4's Table 5). Both readouts come from *one* forward pass per model, so inputs and weights are identical across a pair of rows; "distinct" counts distinct score values over all 4346 held-out steps; Spearman is against the MC continuation value. The row that carries the section is the base model, where the readout is worth −0.0605. Parse rate is defined in Table 3's caption.

model	readout	AUC_{wp} [95%]	step AUC_{glob} [95%]	Spearman	distinct	parse
trained	P(`[[correct]]`)	0.7903 [0.5759, 0.9180]	0.8417 [0.7899, 0.8899]	0.3991	3532	1.0000
trained	argmax	0.5323 [0.4155, 0.6364]	0.5650 [0.5301, 0.6035]	0.1501	2	1.0000
base	P(`[[correct]]`)	0.4435 [0.2652, 0.6140]	0.6264	0.0432	4155	1.0000
base	argmax	0.5040 [0.3762, 0.6164]	0.5973	0.0456	2	1.0000

Figure 2 in Section 1 plots this same ablation as a grouped bar chart, to make the interaction visible at a glance rather than requiring the reader to cross-reference rows.

D.7 Adapter versus merged checkpoint

Every Best-of-N number in this paper is served from the merged bf16 checkpoint, for serving speed. Its held-out AUC has not been separately measured: the two within-problem AUC figures available, 0.7823 and 0.7903, are both re-evaluations of the *same* non-merged rank-256 LoRA adapter, on identical held-out data, through two different scoring code paths (Section 2); their overlapping bootstrap CIs make the gap between them harness noise, not a merging effect. Whether merging changes AUC is therefore unmeasured, not zero – the limitation this creates for the Best-of-N numbers is stated in Section 8.

E The labelling rule in full

E.1 The rule

The rule is stated precisely enough to reimplement from this paragraph alone. Given k successes out of M sampled continuations from a step, compute the Wilson interval for the binomial proportion k/M at the stated confidence level. Label the step positive if the interval's lower bound exceeds $\tau_+ = 0.50$, negative if the interval's upper bound falls below $\tau_- = 0.20$, and abstain (soft-target only, no hard label) otherwise. The soft target used for every row, hard-labelled or not, is a shrunk estimate of k/M rather than the raw ratio; at $k = 0$, $M = 16$ this shrunk estimate is 0.10, not 0, because $k = 0$ in M trials is evidence the true rate is low but is not evidence it is exactly zero. Two threshold facts follow arithmetically from the rule and are worth stating on their own: at $M = 16$, a positive label needs $k \geq 12$ and a negative label needs $k \leq 0$; and at the smaller budget $M = 8$ actually shipped in the reproduction detail files, the Wilson upper bound at $k = 0$ is already 0.3244, above τ_-, so no step sampled only 8 times can ever be labelled negative, regardless of how many of its continuations fail.

E.2 Coverage, computed from the rule

E.3 The measured histogram

Reading the histogram: 20,141 rows sit at $k = 0$ (76.9%), 1948 at $k = 1$ (7.4%), 936 at $k = 2$ (3.6%), 2609 across the middle range $3 \leq k \leq 11$ (10.0%), and 567 at $k \geq 12$ (2.2%). The negative class is exactly the $k = 0$ bucket: the Wilson upper bound at $k = 1$ is already 0.2833, above $\tau_- = 0.20$, so $k = 1$ cannot be labelled negative and $k = 0$ is the only bucket that can be. 21.0% of the corpus falls in the abstention band between the two thresholds and carries no hard label at all. The consequence for a two-class step classifier trained on this corpus is direct: it is trained to separate a single low-k bucket from everything above it, on a distribution that is monotone decreasing rather than bimodal, so the class boundary it

Table 6: Coverage as a function of the labelling budget M, computed analytically from the labelling rule at $\tau_+ = 0.50$ and $\tau_- = 0.20$: this table contains no measurement, only arithmetic consequences of the rule stated above. The in-band coverage column is what Proposition 5 describes.

M	negatives possible	min k for +	max k for −	UB$(0, M)$	coverage in band
16	yes	12	0	0.1936	0.1785

M=16 is the only labelling budget computed in analysis/label_coverage_M16.json as of this snapshot; other M are not fabricated

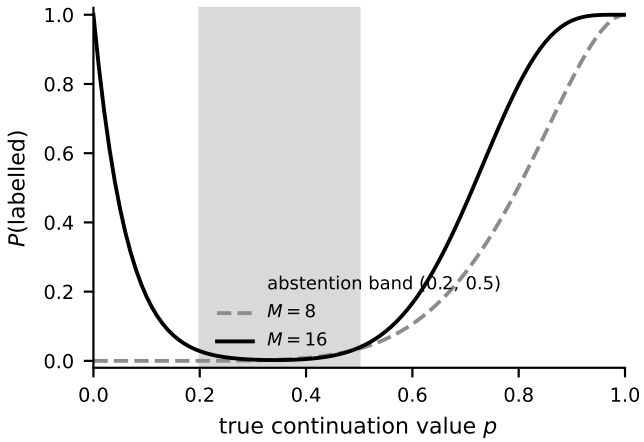


Figure 6: Probability that a step is labelled at all (not abstained on) as a function of its true continuation value p, one curve per M, with the abstention band $(0.20, 0.50)$ shaded. Analytic, not measured: every curve is a closed-form function of the labelling rule and p, with no rollout data behind it.

learns is a threshold on one tail of a skewed distribution rather than a contrast between two well-separated populations.

F The occlusion probe in full

F.1 Construction

The probe replaces the image under a step whose verdict the verifier already got right with one of three counterfactuals – a black canvas, Gaussian noise, or another image drawn from the same visual class – and asks whether the verdict flips. The denominator is 12,026 steps over 4195 problems on which the *verifier's* verdict was correct, not the steps whose MC label happened to be positive. Conditioning on the MC label instead is not a harmless alternative: on any step where the verifier's verdict and the MC label disagree, the verdict cannot flip away from itself, so that step is counted as a non-flip by construction, and every rate computed this way is deflated by exactly the verifier's error rate. The swap counterfactual is seeded deterministically from the problem and step identifiers rather than from a per-process salt, because a salted hash makes a restarted shard draw a different replacement image for the same row than the shard it is resuming, which silently disagrees with rows already written and makes the corpus depend on how many times a shard happened to be restarted.

F.2 Rates

Aggregated over all subsets: black flips 0.3503 [0.3401, 0.3606] of correct verdicts, Gaussian 0.3372 [0.3273, 0.3474], and swap 0.6961 [0.6857, 0.7068] – roughly twice the destruction rate. The single informative quantity is the gap between black and swap, not either rate alone,

Table 7: The measured k histogram over 26,201 labelled steps at $M = 16$, unlike the two tables above. The distribution is monotone decreasing rather than bimodal, and the soft-target column shows what value the two-class threshold rule replaces each bucket with.

k	rows	%	soft target
0	20,141	76.9	0.10
1	1948	7.4	0.15
2	936	3.6	0.20
3–11	2609	10.0	0.25–0.65
≥12	567	2.2	0.70–0.90

Table 8: Occlusion flip rates by benchmark subset, on steps where the verifier's own verdict was correct. Intervals are problem-level bootstrap; subsets are reported separately so the aggregate rates in the text are not read as driven by a single benchmark. Black and Gaussian agree to within 0.0131 with overlapping intervals in every subset, which is the control: two counterfactuals that destroy the image by unrelated means should and do produce indistinguishable flip rates.

subset	steps	black	Gaussian	swap
mathverse	6711	0.3368	0.3406	0.6650
mathvision	3526	0.3279	0.2898	0.6721
mathvista	1581	0.4541	0.4326	0.8722
mmmu_pro	208	0.3798	0.3029	0.7644

because black is out-of-distribution (a verifier can flip on it merely by noticing the image is degenerate) while swap is in-distribution and breaks only the question-specific evidence, keeping spurious priors such as chart-vs-equation intact. That gap, the content deficit black minus swap, is -0.3458 [-0.3559, -0.3358], a negative interval excluding zero: replacing the image with something wrong flips the verdict significantly more often than merely destroying it. Two survival figures follow from the same rates: 65.0% of correct verdicts survive the image being destroyed (averaging black and Gaussian), and 27.1% survive all three corruptions – made, in effect, without the image ever mattering to the verdict.

F.3 Three anticipated regimes and the one observed

The probe was designed to distinguish three regimes: content-sensitive (both swap and black flip the verdict, because the verifier reads what the image shows), presence-sensitive (black flips but swap does not, because the verifier only checks that an image is there), and blind (neither flips). The observed pattern is none of the three: swap flips the verdict about twice as often as black does, which none of the anticipated regimes predicts, since presence-sensitivity requires black to flip *more* than swap and content-sensitivity requires the two to be comparable. The pattern this probe actually measures is that the verifier reacts more to a wrong image than to no image at all. Call this contradiction-sensitive and stop there; it should not be upgraded to a claim of partial grounding, because a verifier that flips on contradiction is not thereby shown to use the image to reach a verdict in the ordinary case, only to notice when a substituted image conflicts with the step text it is also conditioning on. What this bounds: an attribution of the verifier's errors to a perception failure versus a reasoning failure is not interpretable for a verifier whose verdicts do not require the image to support the step in the first place, so Section 8's absence of a perception-versus-reasoning attribution is not a missing experiment this probe could have supplied.

F.4 Why the probe is not used as a filter

Three reasons, and all three are required to justify leaving the corpus unfiltered. First, the probe measures the rationale generator's dependence on the image, while the training target comes from the MC rollout label, not from the rationale; filtering on the flip rate would clean the rationale text without touching the targets, and the rationale is not in the causal

path to the deployed score, which is read from the verdict position and does not depend on any rationale text being generated (Section 4). Second, a filter built on this probe is a second censoring mechanism, of the same species as the abstention band in Section 7: it would select for steps where an off-the-shelf model already carries a strong visual signal, discarding exactly the steps a verifier trained on the filtered corpus would most need to learn about. Third, applying it would confound the training result – a headline AUC measured on a corpus that has already been filtered by an image dependence probe cannot be attributed between the filter and the training objective, and this paper's claim is about the objective. For all three reasons the flip rate is reported here as a result, not applied as preprocessing, and the per-row occlusion flags are retained in the release so that a filtered corpus can be built, and its effect measured, by anyone who disagrees with this choice.

F.5 What the rationales can and cannot be read as

The verdict tag is required as the first token of the reply, so any justification is generated conditioned on a verdict already emitted, not the other way around; in the configuration the scores in this paper come from, only the verdict position's logits are read at inference and no rationale is generated or decoded at all; and 12.1% of the training rationales are exactly one of two label-determined fallback sentences, for which reading the rationale recovers the label the fallback was chosen from and nothing else. The consequence follows once, and does not change with more examples: the rationale can be read as a description of the corpus, but it cannot be read as an audit trail for the verdict, either at training time – because it is conditioned on the verdict rather than the other way around – or at inference time, because in the scoring configuration this paper reports from, it is never produced.

G Baselines and controls in full

G.1 Definitions

Six zero-parameter baselines, each a function of a single scored row and nothing the verifier was trained on: step index (the 0-based position of the candidate step within its completion); prefix word count and prefix character count (the length of everything before the candidate step); candidate step word count; an indicator for whether the candidate step contains a boxed expression; and a constant that ignores the row entirely. The first three – step index, prefix words, prefix characters – are monotone in one another on this corpus, since a prefix's word and character counts both grow with the number of steps that precede it; they are one baseline reported three ways, not three pieces of independent evidence, and they score identically for exactly that reason. The constant baseline ignores the row and returns the same score for every completion, so it cannot rank anything within a problem; it is the empirical check on Corollary 2, which predicts a within-problem AUC of exactly one half for any scorer with no information about the row, and it returns 0.5000, matching the prediction to four decimal places.

G.2 Orientation, and why it is asymmetric

This orientation choice is asymmetric on purpose, and the asymmetry answers a real objection directly. For a nuisance baseline the reported quantity is $\max(\mathrm{AUC}, 1 - \mathrm{AUC})$, because an adversary constructing a baseline has both the feature and its negation available at no cost – a scorer that predicts "earlier is more correct" and one that predicts "later is more correct" are the same one line of code with a sign flipped – so the relevant question is whether the feature carries information at all, not whether the sign an author happened to write down points the right way. Reporting a baseline's raw signed statistic would let a weak baseline look weaker than it is merely because of an arbitrary implementation choice. For the model under test this argument does not apply: its orientation is fixed in advance by the training objective (higher score means more likely correct), and reporting its oriented value would silently convert a model that anti-ranks its own held-out split into a model that appears to rank it, which is not a free option to take. Concretely, the step-index baseline's raw within-problem statistic is 0.3306 and its oriented value is 0.6694. The raw value being below

one half means later steps score as *less* correct under the sign convention 'minus_step_index' happens to use; flipping it to 0.6694 recovers "later steps score as more correct," which is exactly the direction the MC labelling procedure predicts (Section 3), and that external check is why the orientation is not an arbitrary rescue in this particular case.

G.3 Between- and within-problem decomposition

The pooled AUC is the one usually reported, and it hides which of two very different scorers produced it: one that ranks problems by their marginal difficulty (a between-problem effect) and one that ranks completions inside a problem (the within-problem effect the whole paper is about). Decomposing the step-index baseline's AUC into the two shows it is almost entirely the former. Pooled AUC is 0.8417 (124 pairs); the between-problem component, computed over problem-mean scores, is 0.8155 (756 problems); the within-problem component, computed after centring each problem on its own mean, is 0.6342 (124 pairs) – close to chance. The step-index baseline is strong on the axis the label construction predicts (later steps score as more correct, a marginal, per-problem property) and uninformative on the axis this paper measures (which completion, inside a problem, is more likely correct). Reporting only the pooled figure would have let the between-problem effect stand in for a within-problem one.

G.4 The paired bootstrap

Protocol: resample the 39 mixed problems with replacement, recompute both scorers' oriented within-problem AUC on the resampled problem set, and take the difference; repeating this many times gives a bootstrap distribution over the difference, from which the reported 95% interval is the percentile interval and the reported p-value is the two-sided fraction of replicates on the other side of zero from the observed difference. The design is paired rather than a comparison of two independent intervals because the two scorers are evaluated on the same 124 pairs drawn from the same problems, so problem-level noise – a problem that happens to be easy or hard for both scorers at once – is shared and cancels in the difference. The unpaired intervals reported in Section 5 do not get this cancellation, so they are conservative by a wide margin relative to the paired test used here, and a difference that looks ambiguous under two overlapping unpaired intervals can still be resolved by the paired bootstrap on the identical pairs.

Table 9: The full baseline grid: every zero-parameter baseline against the verifier, oriented, on the identical 124 within-problem pairs from 39 mixed held-out problems. AUC_{glob} is the pooled, unoriented statistic reported for reference; the paired column is defined as baseline minus model, so a negative Δ whose interval excludes zero means the verifier significantly outperforms that baseline, and p is the paired bootstrap's two-sided p-value from Section G.4.

scorer	AUC_{wp}	oriented	pairs	AUC_{glob}	paired Δ [95%]	p
candidate has boxed	0.4435	0.5565	124	0.5088	-0.2339 [-0.4153, -0.0402]	0.022
constant	0.5000	0.5000	124	0.5000	–	–
minus candidate words	0.3347	0.6653	124	0.4564	-0.1250 [-0.2851, +0.0443]	0.138
minus prefix chars	0.3306	0.6694	124	0.4140	-0.1210 [-0.2208, -0.0385]	0.007
minus prefix lines	0.3306	0.6694	124	0.4160	-0.1210 [-0.2208, -0.0385]	0.007
minus prefix words	0.3306	0.6694	124	0.4173	-0.1210 [-0.2208, -0.0385]	0.007
minus step index	0.3306	0.6694	124	0.4333	-0.1210 [-0.2208, -0.0385]	0.007

Figure 3 in Section 6 plots this same paired comparison as a forest plot.

G.5 The trajectory-level grid

This is the full grid behind Section 6's summary table: every benchmark against every aggregator, so the claim that aggregator choice does not matter is checkable row by row rather than taken on the strength of a reported range.

Figure 4 in Section 6 shows this same grid as a heatmap.

Table 10: The trajectory-level grid: every benchmark against 6 aggregators, with the length reranker carried in the same row group so the comparison cannot be read as a footnote. Across the grid the verifier's within-problem AUC ranges 0.5069–0.5749 and the length reranker's ranges 0.5075–0.5278; n_{mixed}, the number of problems with at least one correct and one incorrect completion, is the row's true denominator, since it is the only subset on which any reranker can change an answer. Parse rate is the fraction of scored items from which a verdict distribution could be read, i.e. items whose mass on the three verdict tokens clears the scorer's threshold. An item that cannot be parsed is assigned the neutral score and therefore ties with every other such item; a table of scores without a parse rate cannot be read.

benchmark	aggregator	AUC_{wp} [95%]	n_{mixed}	pairs	tie rate	length	parse
mathverse	last	0.5749	–	1,415,901	0.0000	0.5107	–
mathverse	logmean	0.5499	–	1,415,901	0.0000	0.5107	–
mathverse	mean	0.5492	–	1,415,901	0.0000	0.5107	–
mathverse	min	0.5347	–	1,415,901	0.0183	0.5107	–
mathverse	prod	0.5134	–	1,415,901	0.0000	0.5107	–
mathverse	softmin	0.5501	–	1,415,901	0.0000	0.5107	–
mathvision	last	0.5686	–	703,769	0.0000	0.5075	–
mathvision	logmean	0.5596	–	703,769	0.0000	0.5075	–
mathvision	mean	0.5596	–	703,769	0.0000	0.5075	–
mathvision	min	0.5360	–	703,769	0.0122	0.5075	–
mathvision	prod	0.5249	–	703,769	0.0000	0.5075	–
mathvision	softmin	0.5592	–	703,769	0.0000	0.5075	–
mathvista	last	0.5477	–	342,818	0.0100	0.5131	–
mathvista	logmean	0.5435	–	342,818	0.0050	0.5131	–
mathvista	mean	0.5434	–	342,818	0.0050	0.5131	–
mathvista	min	0.5127	–	342,818	0.0680	0.5131	–
mathvista	prod	0.5069	–	342,818	0.0070	0.5131	–
mathvista	softmin	0.5438	–	342,818	0.0060	0.5131	–
mmmu_pro	last	0.5180	–	36,846	0.0000	0.5278	–
mmmu_pro	logmean	0.5316	–	36,846	0.0000	0.5278	–
mmmu_pro	mean	0.5300	–	36,846	0.0000	0.5278	–
mmmu_pro	min	0.5156	–	36,846	0.0064	0.5278	–
mmmu_pro	prod	0.5086	–	36,846	0.0035	0.5278	–
mmmu_pro	softmin	0.5324	–	36,846	0.0000	0.5278	–

selector_summary.json carries no per-cell bootstrap CI, mixed-problem count, or parse rate at trajectory level (0 of its rows have a parse_rate key); see within_problem.json and occlusion_flip_rates.json for the step-level equivalents

G.6 An externally-trained verifier, on the same cache

Reviewers of an earlier draft asked the question Section 6 now answers in the body: is the trajectory-level null a property of this project's own checkpoint, or of multimodal process-reward verifiers more broadly? Table 11 answers it for one externally-trained candidate. VisualPRM-8B (Wang et al., 2025) was served through the same vLLM pipeline as every other scorer in this paper (its own +/− verdict tokens probed rather than assumed, per its own protocol), then scored on the identical four rollout caches Table 1 uses, with the identical softmin aggregator, `within_problem.pair_counts` pooling, and a 2000-resample problem-level bootstrap—the same recipe as every other AUC_{wp} in this paper. The pair counts in the last column match Table 1's PRM row exactly benchmark-for-benchmark, which is the check that the two verifiers were scored on the same problem/completion universe rather than different samples.

VisualPRM-8B's interval excludes 0.5 on every benchmark: unlike this project's own checkpoint, it carries real (if still modest—well short of the training-domain step-level ceiling 0.8435 this paper's own labels support) within-problem ranking signal. It beats the length reranker everywhere and this paper's own softmin PRM point estimate on three of four benchmarks, falling below it only on MathVerse. Read plainly: the null reported in Section 6 does not generalise to every multimodal verifier on this cache, so it is evidence about this

checkpoint, this training run, and/or this protocol—not a property of multimodal process supervision that this measurement can localise further, and not license to claim BoN selection with *either* verifier is reliable here.

Table 11: VisualPRM-8B (Wang et al., 2025), an externally trained multimodal PRM this project did not train, scored on the same four rollout caches as Table 1 under the identical softmin aggregator and bootstrap recipe. "ours (softmin)" and "length" repeat the corresponding cells of Table 1 for comparison. n_{mixed} and pairs are VisualPRM-8B's own denominators; they match Table 1's PRM row exactly, confirming the same problem/completion universe. Parse rate is defined in Table 3's caption.

benchmark	VisualPRM-8B AUC_{wp} [95%]	ours (softmin)	length	n_{mixed}	pairs	parse
MathVista	0.6184 [0.5966, 0.6434]	0.5438	0.5131	610	342,818	1.0000
MathVerse	0.5233 [0.5151, 0.5316]	0.5501	0.5107	2512	1,415,901	1.0000
MathVision	0.5849 [0.5724, 0.5970]	0.5592	0.5075	1437	703,769	1.0000
MMMU-Pro	0.5724 [0.5072, 0.6364]	0.5324	0.5278	81	36,846	1.0000

Figure 5b in Section 6 draws the same comparison as a grouped bar chart, so the asymmetry the numbers above describe – both in-house scorers sitting on the chance line while VisualPRM-8B clears it on all four benchmarks – is visible at a glance rather than read off six columns.

G.7 How the field reports step-level verifiers

Table 12 is the property table behind Section 2's positioning claim: read from each paper's own tables, not a summary of this paper's own results.

Table 12: How step-level verifiers are reported, read from each paper's own tables: PRM800K (Lightman et al., 2024), Math-Shepherd (Wang et al., 2024b), OmegaPRM (Luo et al., 2024), VisualPRM (Wang et al., 2025), ProcessBench (Zheng et al., 2024), PRMBench (Song et al., 2025). "Chains/problem" (distinct labelled solution chains per problem) governs positionality (Section 5). Dash: not reported, not necessarily bad; "part.": a within-chain quantity is reported but not a within-problem ranking metric. Last row: this paper; its chains/problem entry is a weakness, discussed in Section 2, not a claim.

system	step labels	chains /problem	headline metric	within-prob. metric	nuisance baseline	parse
PRM800K	human	> 1	BoN	no	no	–
Math-Shepherd	MC	> 1	BoN	no	no	–
OmegaPRM	tree/MC	tree	BoN	no	no	–
VisualPRM	MC	> 1	BoN	no	no	–
ProcessBench	human	1	error loc.	part.	no	–
PRMBench	synthetic	> 1	multi-axis	part.	no	–
this paper	MC (soft)	1	AUC_{wp}	yes	position, length	1.0000

H Provenance of every number

H.1 The rule, and how it is enforced

State it plainly: the body of this paper contains no hand-typed number. Every figure that appears in running text or a table is a macro, generated by `scripts/make_numbers.py` from a result JSON file under `results/`, at the same time as the mapping in Table 13 below. A macro with no backing file is not emitted by the generator; a macro the body uses that the generator did not emit is an undefined control sequence, and an undefined control sequence fails the LaTeX build outright, on this file and every file that cites it. This is why there are no fallback values anywhere in the source: a fallback value would let a missing measurement typeset as a plausible-looking number instead of stopping the build, which

is precisely the silent failure this paper's own methodology sections argue against in the pipeline that produced the submitted verifier.

H.2 Pipeline stages and which of them ran

The stages that ran, in order, each with the script that ran it and the result file it produced:

- **Rollout generation.** `scripts/box/01_rollouts.sh` samples N=64 completions per problem, writing `data/rollouts/` and `data/rollouts_train/`; audited by `src.eval.audit_rollouts` into `analysis/rollout_audit.json`.
- **MC labelling.** `scripts/box/02_mclabel.sh` applies the Wilson verdict rule to the rollout cache, writing `data/labeled/` (26,201 labelled steps over 4640 problems) and `data/prm_train/corpus_stats.json`.
- **Rationale generation and the occlusion probe.** `scripts/box/03_rationale.sh` writes `data/prm_train/`, `data/verifications/`, and `occlusion_flip_rates.json` (App. F).
- **Verifier training.** `scripts/box/04_train_prm.sh` writes `checkpoints/*/held_out_verdict_report.json`; the readout ablation that isolates the repair's contribution (`src.eval.readout_ablation`) writes `analysis/readout_ablation.json` and `readout_scores.*.jsonl`.
- **Held-out evaluation.** The step-level baselines and paired bootstrap (`src.eval.step_baselines`) write `analysis/step_baselines_final.json`, this appendix's Table 9.
- **Best-of-N evaluation.** `scripts/box/05_bon_eval.sh` and `05b_indomain_bon.sh` write `analysis/step_eval.json` and `analysis/indomain_bon.json`.
- **Rescoring through the corrected instrument.** `scripts/box/05c_bon_faithful.sh` re-serves the merged checkpoint with the four scoring defects fixed at once (image sent, chat template applied, no injected question field, exact verdict-token readout), writing `results/scored/05c_faithful.*/` and `analysis/selectors_faithful.*.json`, the source of Table 10.

Section 8 depends on the following list being complete, so it is stated here in the same terms it uses there: the stages that did *not* run are the head-to-head against an external multimodal process reward model (`scripts/box/06_headtohead.sh`), cross-generator transfer (`scripts/box/07_cross_family.sh`), the reinforcement-learning arms (`scripts/box/09_rlvr.sh` and `10_rlvr_eval.sh`), the attribution analysis of perception versus reasoning (not scripted; bounded in advance by the occlusion probe, App. F), and the human audit of step labels (`src.eval.human_audit`, implemented and tested but never run against a collected rating). The GPU box these stages would have run on was destroyed before they were, so these are gaps in this artifact, not future work.

H.3 Macro provenance

Table 13: Every macro this paper's body and appendices actually cite, its backing result file, and the JSON key path (or, for a derived quantity, the expression over key paths) it was computed from. Generated by the same invocation of `scripts/make_numbers.py` that writes `tex/numbers.tex`, from the same internal mapping, so this table cannot disagree with the body: a row here that named the wrong file or key would be a wrong number there. Restricted to the 262 macros this paper cites, out of several thousand the underlying pipeline computes in total (exhaustive benchmark × aggregator × N × selector sweeps the prose never draws on); listing every uncited combination would not make any claim in this paper more checkable, only longer.

macro	result file	key
`NumBaselineGridTableBody`	analysis/step_baselines_final.json	baselines.* + paired_vs_baselines.*

Table 13 (cont.)

macro	result file	key
NumBonAggregators	analysis/selector_summary.json	count(distinct row.aggregator) over rows for the 4 non-heldout BoN caches rollouts_mathverse, rollouts_mathvision, rollouts_mathvista, rollouts_mmmu_pro
NumBonAucMax	analysis/selector_summary.json	max(auc) over the 4 non-heldout BoN caches rollouts_mathverse, rollouts_mathvision, rollouts_mathvista, rollouts_mmmu_pro = [1].auc
NumBonAucMin	analysis/selector_summary.json	min(auc) over the 4 non-heldout BoN caches rollouts_mathverse, rollouts_mathvision, rollouts_mathvista, rollouts_mmmu_pro = [111].auc
NumBonLengthAucMax	analysis/selector_summary.json	max(auc) over the 4 non-heldout BoN caches rollouts_mathverse, rollouts_mathvision, rollouts_mathvista, rollouts_mmmu_pro = [150].auc
NumBonLengthAucMin	analysis/selector_summary.json	min(auc) over the 4 non-heldout BoN caches rollouts_mathverse, rollouts_mathvision, rollouts_mathvista, rollouts_mmmu_pro = [30].auc
NumBonMathverseSoftmin SelectorMajorityAtNEightAcc	analysis/selectors.rollouts_mathverse.softmin.json	sweep.8.majority.acc
NumBonMathverseSoftmin SelectorPrmWmajAtNEightAcc	analysis/selectors.rollouts_mathverse.softmin.json	sweep.8.wmaj:prm.acc
NumBonMathvisionSoftmin SelectorMajorityAtNEightDelta VsTopOne	analysis/selectors.rollouts_mathvision.softmin.json	sweep.8.majority.acc - sweep.8.first.acc
NumBonMathvisionSoftmin SelectorPrmWmajAtNEightDelta VsMajority	analysis/selectors.rollouts_mathvision.softmin.json	sweep.8.wmaj:prm.delta_vs_majority
NumBonMathvisionSoftmin SelectorPrmWmajAtNEightDelta VsTopOne	analysis/selectors.rollouts_mathvision.softmin.json	sweep.8.wmaj:prm.acc - sweep.8.first.acc
NumBonMathvistaSoftmin SelectorMajorityAtNSixtyFour Acc	analysis/selectors.rollouts_mathvista.softmin.json	sweep.64.majority.acc
NumBonMathvistaSoftmin SelectorOracleAtNSixtyFourAcc	analysis/selectors.rollouts_mathvista.softmin.json	sweep.64.oracle.acc
NumBonMathvistaSoftmin SelectorRandomAtNSixtyFourAcc	analysis/selectors.rollouts_mathvista.softmin.json	sweep.64.random.acc
NumBonMmmuProAucMax	analysis/selector_summary.json	max(auc) over the aggregators swept on the rollouts_mmmu_pro cache = [176].auc
NumBonMmmuProAucMin	analysis/selector_summary.json	min(auc) over the aggregators swept on the rollouts_mmmu_pro cache = [171].auc
NumBonRolloutsMathverse SoftminLengthAtNSixtyFourAuc	analysis/selector_summary.json	[25].auc
NumBonRolloutsMathverse SoftminPrmAtNSixtyFourAuc	analysis/selector_summary.json	[26].auc
NumBonRolloutsMathvision SoftminLengthAtNSixtyFourAuc	analysis/selector_summary.json	[55].auc
NumBonRolloutsMathvision SoftminPrmAtNSixtyFourAuc	analysis/selector_summary.json	[56].auc
NumBonRolloutsMathvista SoftminLengthAtNSixtyFourAuc	analysis/selector_summary.json	[115].auc
NumBonRolloutsMathvista SoftminPrmAtNSixtyFourAuc	analysis/selector_summary.json	[116].auc
NumBonRolloutsMmmuProSoftmin LengthAtNSixtyFourAuc	analysis/selector_summary.json	[175].auc
NumBonRolloutsMmmuProSoftmin PrmAtNSixtyFourAuc	analysis/selector_summary.json	[176].auc
NumCZeroGatePrmHardDistinct StepVectors	analysis/c0_gate.json	sum(rows[mode=hard].distinct_step_vectors)
NumCZeroGatePrmHardTieFrac	analysis/c0_gate.json	sum(rows[mode=hard].argmax_tie_frac * mixed_problems) / sum(rows[mode=hard].mixed_problems)
NumCZeroGatePrmProbDistinct StepVectors	analysis/c0_gate.json	sum(rows[mode=prob].distinct_step_vectors)
NumCZeroGatePrmProbTieFrac	analysis/c0_gate.json	sum(rows[mode=prob].argmax_tie_frac * mixed_problems) / sum(rows[mode=prob].mixed_problems)
NumCorpusAbstainBandFracPct	analysis/corpus_stats.json	100 * sum(k_histogram[1..11]) / n_rows
NumCorpusBoilerplateFracPct	analysis/corpus_stats.json	100 * boilerplate_frac
NumCorpusKbucketGeTwelve	analysis/corpus_stats.json	sum(k_histogram[ge_twelve])
NumCorpusKbucketGeTwelveFrac Pct	analysis/corpus_stats.json	100 * sum(k_histogram[ge_twelve]) / n_rows
NumCorpusKbucketOne	analysis/corpus_stats.json	sum(k_histogram[one])
NumCorpusKbucketOneFracPct	analysis/corpus_stats.json	100 * sum(k_histogram[one]) / n_rows
NumCorpusKbucketThreeToEleven	analysis/corpus_stats.json	sum(k_histogram[three_to_eleven])

Table 13 (cont.)

macro	result file	key
NumCorpusKbucketThreeToElevenFracPct	analysis/corpus_stats.json	100 * sum(k_histogram[three_to_eleven]) / n_rows
NumCorpusKbucketTwo	analysis/corpus_stats.json	sum(k_histogram[two])
NumCorpusKbucketTwoFracPct	analysis/corpus_stats.json	100 * sum(k_histogram[two]) / n_rows
NumCorpusKbucketZero	analysis/corpus_stats.json	sum(k_histogram[zero])
NumCorpusKbucketZeroFracPct	analysis/corpus_stats.json	100 * sum(k_histogram[zero]) / n_rows
NumCorpusMcSamples	analysis/corpus_stats.json	realized_M.16 (only key)
NumCorpusProblems	analysis/corpus_stats.json	n_problems
NumCorpusRows	analysis/corpus_stats.json	n_rows
NumCorpusSoftTargetAtKZero	analysis/corpus_stats.json	soft_target_at_k_zero
NumCorpusSoftTargetFracPct	analysis/corpus_stats.json	100 * soft_target_frac
NumCoverageMEightWilsonUpperAtZero	analysis/label_coverage_M8.json	budgets.8.wilson_upper_at_zero
NumCoverageMSixteenMaxKNegative	analysis/label_coverage_M16.json	budgets.16.max_k_negative
NumCoverageMSixteenMinKPositive	analysis/label_coverage_M16.json	budgets.16.min_k_positive
NumCoverageMSixteenWilsonUpperAtOne	analysis/label_coverage_M16.json	wilson_upper(k=1, n=budgets.16.m, z) [computed, not a field: matches wilson_ci in src/data/mc_label.py]
NumCoverageMSixteenWilsonUpperAtZero	analysis/label_coverage_M16.json	budgets.16.wilson_upper_at_zero
NumCoverageShippedMcSamples	analysis/label_coverage_M8.json	budgets.8.m
NumCoverageTauNeg	analysis/label_coverage_M16.json	tau_neg
NumCoverageTauPos	analysis/label_coverage_M16.json	tau_pos
NumExtprmAucMax	analysis/external_verifier_visualprm_mathverse.json, analysis/external_verifier_visualprm_mathvision.json, analysis/external_verifier_visualprm_mathvista.json, analysis/external_verifier_visualprm_mmmu_pro.json	max(auc) over 4 benchmarks = mathvista.auc
NumExtprmAucMin	analysis/external_verifier_visualprm_mathverse.json, analysis/external_verifier_visualprm_mathvision.json, analysis/external_verifier_visualprm_mathvista.json, analysis/external_verifier_visualprm_mmmu_pro.json	min(auc) over 4 benchmarks = mathverse.auc
NumExtprmMathverseAuc	analysis/external_verifier_visualprm_mathverse.json	auc
NumExtprmMathverseAucCiHi	analysis/external_verifier_visualprm_mathverse.json	ci_lo/ci_hi
NumExtprmMathverseAucCiLo	analysis/external_verifier_visualprm_mathverse.json	ci_lo/ci_hi
NumExtprmMathverseAucN	analysis/external_verifier_visualprm_mathverse.json	n_pairs
NumExtprmMathverseParseRate	analysis/external_verifier_visualprm_mathverse.json	parse_rate
NumExtprmMathverseProblems	analysis/external_verifier_visualprm_mathverse.json	n_problems
NumExtprmMathvisionAuc	analysis/external_verifier_visualprm_mathvision.json	auc
NumExtprmMathvisionAucCiHi	analysis/external_verifier_visualprm_mathvision.json	ci_lo/ci_hi
NumExtprmMathvisionAucCiLo	analysis/external_verifier_visualprm_mathvision.json	ci_lo/ci_hi
NumExtprmMathvisionAucN	analysis/external_verifier_visualprm_mathvision.json	n_pairs
NumExtprmMathvisionParseRate	analysis/external_verifier_visualprm_mathvision.json	parse_rate
NumExtprmMathvisionProblems	analysis/external_verifier_visualprm_mathvision.json	n_problems
NumExtprmMathvistaAuc	analysis/external_verifier_visualprm_mathvista.json	auc
NumExtprmMathvistaAucCiHi	analysis/external_verifier_visualprm_mathvista.json	ci_lo/ci_hi
NumExtprmMathvistaAucCiLo	analysis/external_verifier_visualprm_mathvista.json	ci_lo/ci_hi
NumExtprmMathvistaAucN	analysis/external_verifier_visualprm_mathvista.json	n_pairs
NumExtprmMathvistaParseRate	analysis/external_verifier_visualprm_mathvista.json	parse_rate

Table 13 (cont.)

macro	result file	key
NumExtprmMathvistaProblems	analysis/external_verifier_visualprm_mathvista.json	n_problems
NumExtprmMmmuProAuc	analysis/external_verifier_visualprm_mmmu_pro.json	auc
NumExtprmMmmuProAucCiHi	analysis/external_verifier_visualprm_mmmu_pro.json	ci_lo/ci_hi
NumExtprmMmmuProAucCiLo	analysis/external_verifier_visualprm_mmmu_pro.json	ci_lo/ci_hi
NumExtprmMmmuProAucN	analysis/external_verifier_visualprm_mmmu_pro.json	n_pairs
NumExtprmMmmuProParseRate	analysis/external_verifier_visualprm_mmmu_pro.json	parse_rate
NumExtprmMmmuProProblems	analysis/external_verifier_visualprm_mmmu_pro.json	n_problems
NumGraderImpossibleTrueNoAnswer	analysis/grader_audit.json	graded_true_no_answer
NumGraderPathBoxedFracPct	analysis/grader_audit.json	100 * extraction_path.boxed / n_completions_graded
NumGraderPathLastNumberFracPct	analysis/grader_audit.json	100 * extraction_path.last_number / n_completions_graded
NumGraderPathNoneFracPct	analysis/grader_audit.json	100 * extraction_path.none / n_completions_graded
NumHeldoutLoraRank	analysis/prm_adapter_config.json	r
NumHeldoutMixedProblems	analysis/held_out_verdict_report.json	within_problem_mixed_problems
NumHeldoutNegative	analysis/held_out_verdict_report.json	n_negative
NumHeldoutPositive	analysis/held_out_verdict_report.json	n_positive
NumHeldoutProblems	analysis/held_out_verdict_report.json	n_problems
NumHeldoutScored	analysis/held_out_verdict_report.json	n_scored
NumHeldoutSoftTargetOnly	analysis/held_out_verdict_report.json	n_scored - n_hard_labelled
NumHeldoutSplitFracPct	analysis/held_out_verdict_report.json + analysis/corpus_stats.json	100 * held_out_verdict_report.json::n_problems / corpus_stats.json::n_problems
NumHeldoutWithinAuc	analysis/held_out_verdict_report.json	within_problem_auc
NumHeldoutWithinAucN	analysis/held_out_verdict_report.json	within_problem_pairs
NumKHistTableBody	analysis/corpus_stats.json	k_histogram
NumOcclusionBlackGaussianGap	occlusion_flip_rates.json	pooled.black_gaussian_gap
NumOcclusionByBenchTableBody	occlusion_flip_rates.json	per_source.*
NumOcclusionContentDeficit	occlusion_flip_rates.json	pooled.content_deficit
NumOcclusionContentDeficitCiHi	occlusion_flip_rates.json	pooled.content_deficit_ci
NumOcclusionContentDeficitCiLo	occlusion_flip_rates.json	pooled.content_deficit_ci
NumOcclusionFlipBlack	occlusion_flip_rates.json	pooled.flip_rate_black
NumOcclusionFlipBlackCiHi	occlusion_flip_rates.json	pooled.flip_rate_black_ci
NumOcclusionFlipBlackCiLo	occlusion_flip_rates.json	pooled.flip_rate_black_ci
NumOcclusionFlipGaussian	occlusion_flip_rates.json	pooled.flip_rate_gaussian
NumOcclusionFlipGaussianCiHi	occlusion_flip_rates.json	pooled.flip_rate_gaussian_ci
NumOcclusionFlipGaussianCiLo	occlusion_flip_rates.json	pooled.flip_rate_gaussian_ci
NumOcclusionFlipSwap	occlusion_flip_rates.json	pooled.flip_rate_swap
NumOcclusionFlipSwapCiHi	occlusion_flip_rates.json	pooled.flip_rate_swap_ci
NumOcclusionFlipSwapCiLo	occlusion_flip_rates.json	pooled.flip_rate_swap_ci
NumOcclusionProblems	occlusion_flip_rates.json	pooled.n_problems
NumOcclusionSurvivesAllThreePct	occlusion_flip_rates.json	100 * pooled.survives_all_three
NumOcclusionSurvivesImageDestructionPct	occlusion_flip_rates.json	100 * pooled.survives_image_destruction
NumOcclusionVerifierCorrectSteps	occlusion_flip_rates.json	pooled.n_verifier_correct
NumReadoutBaseHardParseRate	analysis/readout_ablation_final.json	models.base.parse_rate
NumReadoutBaseHardSpearman	analysis/readout_ablation_final.json	models.base.hard.spearman_p_hat
NumReadoutBaseHardStepAuc	analysis/readout_ablation_final.json	models.base.hard.step_auc
NumReadoutBaseHardWithinAuc	analysis/readout_ablation_final.json	models.base.hard.within_problem_auc
NumReadoutBaseHardWithinAucCiHi	analysis/readout_ablation_final.json	models.base.hard.within_problem_auc_ci

Table 13 (cont.)

macro	result file	key
NumReadoutBaseHardWithinAucCiLo	analysis/readout_ablation_final.json	models.base.hard.within_problem_auc_ci
NumReadoutBaseProbDistinctValues	analysis/readout_ablation_final.json	models.base.prob.n_distinct_exact
NumReadoutBaseProbParseRate	analysis/readout_ablation_final.json	models.base.parse_rate
NumReadoutBaseProbSpearman	analysis/readout_ablation_final.json	models.base.prob.spearman_p_hat
NumReadoutBaseProbStepAuc	analysis/readout_ablation_final.json	models.base.prob.step_auc
NumReadoutBaseProbWithinAuc	analysis/readout_ablation_final.json	models.base.prob.within_problem_auc
NumReadoutBaseProbWithinAucCiHi	analysis/readout_ablation_final.json	models.base.prob.within_problem_auc_ci
NumReadoutBaseProbWithinAucCiLo	analysis/readout_ablation_final.json	models.base.prob.within_problem_auc_ci
NumReadoutFinalTrainedHardDistinctValues	analysis/readout_ablation_final.json	models.trained.hard.n_distinct_exact
NumReadoutFinalTrainedHardSpearman	analysis/readout_ablation_final.json	models.trained.hard.spearman_p_hat
NumReadoutFinalTrainedHardStepAuc	analysis/readout_ablation_final.json	models.trained.hard.step_auc
NumReadoutFinalTrainedHardStepAucCiHi	analysis/readout_ablation_final.json	models.trained.hard.step_auc_ci
NumReadoutFinalTrainedHardStepAucCiLo	analysis/readout_ablation_final.json	models.trained.hard.step_auc_ci
NumReadoutFinalTrainedHardWithinAuc	analysis/readout_ablation_final.json	models.trained.hard.within_problem_auc
NumReadoutFinalTrainedHardWithinAucCiHi	analysis/readout_ablation_final.json	models.trained.hard.within_problem_auc_ci
NumReadoutFinalTrainedHardWithinAucCiLo	analysis/readout_ablation_final.json	models.trained.hard.within_problem_auc_ci
NumReadoutFinalTrainedParseRate	analysis/readout_ablation_final.json	models.trained.parse_rate
NumReadoutFinalTrainedProbDistinctValues	analysis/readout_ablation_final.json	models.trained.prob.n_distinct_exact
NumReadoutFinalTrainedProbFracAtModePct	analysis/readout_ablation_final.json	100 * models.trained.prob.frac_at_mode
NumReadoutFinalTrainedProbSpearman	analysis/readout_ablation_final.json	models.trained.prob.spearman_p_hat
NumReadoutFinalTrainedProbSpearmanCiHi	analysis/readout_ablation_final.json	models.trained.prob.spearman_p_hat_ci
NumReadoutFinalTrainedProbSpearmanCiLo	analysis/readout_ablation_final.json	models.trained.prob.spearman_p_hat_ci
NumReadoutFinalTrainedProbStepAuc	analysis/readout_ablation_final.json	models.trained.prob.step_auc
NumReadoutFinalTrainedProbStepAucCiHi	analysis/readout_ablation_final.json	models.trained.prob.step_auc_ci
NumReadoutFinalTrainedProbStepAucCiLo	analysis/readout_ablation_final.json	models.trained.prob.step_auc_ci
NumReadoutFinalTrainedProbWithinAuc	analysis/readout_ablation_final.json	models.trained.prob.within_problem_auc
NumReadoutFinalTrainedProbWithinAucCiHi	analysis/readout_ablation_final.json	models.trained.prob.within_problem_auc_ci
NumReadoutFinalTrainedProbWithinAucCiLo	analysis/readout_ablation_final.json	models.trained.prob.within_problem_auc_ci
NumReadoutGapBaseWithinAuc	analysis/readout_ablation_final.json	models.base.prob.within_problem_auc - models.base.hard.within_problem_auc
NumReadoutGapTrainedStepAuc	analysis/readout_ablation_final.json	models.trained.prob.step_auc - models.trained.hard.step_auc
NumReadoutGapTrainedWithinAuc	analysis/readout_ablation_final.json	models.trained.prob.within_problem_auc - models.trained.hard.within_problem_auc
NumReadoutTrainedHardParseRate	analysis/readout_ablation_final.json	models.trained.parse_rate
NumReadoutTrainedProbParseRate	analysis/readout_ablation_final.json	models.trained.parse_rate
NumReadoutTrainedProbVerdictMass	analysis/readout_ablation_final.json	models.trained.mean_verdict_mass
NumReadoutTrainingGapWithinAuc	analysis/readout_ablation_final.json	models.trained.prob.within_problem_auc - models.base.prob.within_problem_auc
NumReproTableBody	analysis/within_problem.json	¡all cells¿
NumRlvrAlpha	analysis/rlvr_diagnostics.json	alpha
NumRlvrDownstreamEvalStatus	analysis/rlvr_diagnostics.json	downstream_eval.status
NumRlvrGroupSizeTarget	analysis/rlvr_diagnostics.json	group_size_target
NumRlvrGrpoOutcomeOnlyDegenerateGroups	analysis/rlvr_diagnostics.json	arms.grpo_outcome_only.degenerate_groups
NumRlvrGrpoOutcomeOnlyGroups	analysis/rlvr_diagnostics.json	arms.grpo_outcome_only.n_groups

Table 13 (cont.)

macro	result file	key
NumRlvrGrpoOutcomeOnlyMeanAdvantageGap	analysis/rlvr_diagnostics.json	arms.grpo_outcome_only.mean_advantage_gap
NumRlvrGrpoOutcomeOnlyRescuedFracOfDegeneratePct	analysis/rlvr_diagnostics.json	100 * arms.grpo_outcome_only.rescued_frac_of_degenerate
NumRlvrGrpoOutcomeOnlyRescuedGroups	analysis/rlvr_diagnostics.json	arms.grpo_outcome_only.rescued_groups
NumRlvrGrpoOutcomeOnlySamples	analysis/rlvr_diagnostics.json	arms.grpo_outcome_only.n_samples
NumRlvrGrpoOutcomeOnlyZeroDispersionFracPct	analysis/rlvr_diagnostics.json	100 * arms.grpo_outcome_only.zero_dispersion_group_frac
NumRlvrGrpoPrmHardDegenerateGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_hard.degenerate_groups
NumRlvrGrpoPrmHardGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_hard.n_groups
NumRlvrGrpoPrmHardMaxAdvantageGap	analysis/rlvr_diagnostics.json	arms.grpo_prm_hard.max_advantage_gap
NumRlvrGrpoPrmHardMeanAdvantageGap	analysis/rlvr_diagnostics.json	arms.grpo_prm_hard.mean_advantage_gap
NumRlvrGrpoPrmHardRescuedFracOfDegeneratePct	analysis/rlvr_diagnostics.json	100 * arms.grpo_prm_hard.rescued_frac_of_degenerate
NumRlvrGrpoPrmHardRescuedGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_hard.rescued_groups
NumRlvrGrpoPrmHardSamples	analysis/rlvr_diagnostics.json	arms.grpo_prm_hard.n_samples
NumRlvrGrpoPrmHardZeroDispersionFracPct	analysis/rlvr_diagnostics.json	100 * arms.grpo_prm_hard.zero_dispersion_group_frac
NumRlvrGrpoPrmHardZeroDispersionGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_hard.zero_dispersion_groups
NumRlvrGrpoPrmProbDegenerateGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_prob.degenerate_groups
NumRlvrGrpoPrmProbGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_prob.n_groups
NumRlvrGrpoPrmProbMeanAdvantageGap	analysis/rlvr_diagnostics.json	arms.grpo_prm_prob.mean_advantage_gap
NumRlvrGrpoPrmProbRescuedFracOfDegeneratePct	analysis/rlvr_diagnostics.json	100 * arms.grpo_prm_prob.rescued_frac_of_degenerate
NumRlvrGrpoPrmProbRescuedGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_prob.rescued_groups
NumRlvrGrpoPrmProbSamples	analysis/rlvr_diagnostics.json	arms.grpo_prm_prob.n_samples
NumRlvrGrpoPrmProbZeroDispersionFracPct	analysis/rlvr_diagnostics.json	100 * arms.grpo_prm_prob.zero_dispersion_group_frac
NumRlvrGrpoPrmProbZeroDispersionGroups	analysis/rlvr_diagnostics.json	arms.grpo_prm_prob.zero_dispersion_groups
NumRolloutAllProblems	analysis/rollout_audit.json	sum(benchmarks.*.n_problems)
NumRolloutExpectN	analysis/rollout_audit.json	expect_n
NumRolloutMathverseMixed	analysis/rollout_audit.json	benchmarks.mathverse.mixed
NumRolloutMathverseProblems	analysis/rollout_audit.json	benchmarks.mathverse.n_problems
NumRolloutMathvisionMixed	analysis/rollout_audit.json	benchmarks.mathvision.mixed
NumRolloutMathvisionProblems	analysis/rollout_audit.json	benchmarks.mathvision.n_problems
NumRolloutMathvistaDistinctAnswers	analysis/rollout_audit.json	benchmarks.mathvista.mean_distinct_answers
NumRolloutMathvistaMixed	analysis/rollout_audit.json	benchmarks.mathvista.mixed
NumRolloutMathvistaProblems	analysis/rollout_audit.json	benchmarks.mathvista.n_problems
NumRolloutMmmuProDistinctAnswers	analysis/rollout_audit.json	benchmarks.mmmu_pro.mean_distinct_answers
NumRolloutMmmuProMixed	analysis/rollout_audit.json	benchmarks.mmmu_pro.mixed
NumRolloutMmmuProMixedFracPct	analysis/rollout_audit.json	100 * benchmarks.mmmu_pro.mixed_frac
NumRolloutMmmuProProblems	analysis/rollout_audit.json	benchmarks.mmmu_pro.n_problems
NumRolloutShards	analysis/rollout_audit.json	benchmarks.*.n_shards (uniform across every benchmark)
NumRolloutUsableMixed	analysis/rollout_audit.json	sum(benchmarks.*.mixed where mixed_frac ¿= 0.05) over mathverse, mathvision, mathvista
NumSeedTableBody	(absent)	arms*.json
NumStepFinalBaselineCandidateHasBoxedWithinAucOriented	analysis/step_baselines_final.json	baselines.candidate_has_boxed.within_problem_auc_oriented
NumStepFinalBaselineConstantWithinAucOriented	analysis/step_baselines_final.json	baselines.constant.within_problem_auc_oriented
NumStepFinalBaselineMinusCandidateWordsWithinAucOriented	analysis/step_baselines_final.json	baselines.minus_candidate_words.within_problem_auc_oriented
NumStepFinalBaselineMinusPrefixCharsWithinAucOriented	analysis/step_baselines_final.json	baselines.minus_prefix_chars.within_problem_auc_oriented
NumStepFinalBaselineMinusPrefixWordsWithinAucOriented	analysis/step_baselines_final.json	baselines.minus_prefix_words.within_problem_auc_oriented
NumStepFinalBaselineMinusStepIndexWithinAuc	analysis/step_baselines_final.json	baselines.minus_step_index.within_problem_auc
NumStepFinalBaselineMinusStepIndexWithinAucOriented	analysis/step_baselines_final.json	baselines.minus_step_index.within_problem_auc_oriented
NumStepFinalDecompBetweenAuc	analysis/step_baselines_final.json	decomposition.auc_between

Table 13 (cont.)

macro	result file	key
NumStepFinalDecompBetweenAucN	analysis/step_baselines_final.json	decomposition.n_problems_between
NumStepFinalDecompPooledAucOriented	analysis/step_baselines_final.json	decomposition.auc_pooled_oriented
NumStepFinalDecompPooledAucOrientedN	analysis/step_baselines_final.json	decomposition.n_within_pairs
NumStepFinalDecompWithinCentredAuc	analysis/step_baselines_final.json	decomposition.auc_within_centred
NumStepFinalDecompWithinCentredAucN	analysis/step_baselines_final.json	decomposition.n_within_pairs
NumStepFinalPairedCandidateHasBoxedP	analysis/step_baselines_final.json	paired_vs_baselines.candidate_has_boxed.p_two_sided
NumStepFinalPairedMinusCandidateWordsDiff	analysis/step_baselines_final.json	paired_vs_baselines.minus_candidate_words.diff_baseline_minus_model
NumStepFinalPairedMinusCandidateWordsP	analysis/step_baselines_final.json	paired_vs_baselines.minus_candidate_words.p_two_sided
NumStepFinalPairedMinusPrefixWordsP	analysis/step_baselines_final.json	paired_vs_baselines.minus_prefix_words.p_two_sided
NumStepFinalPairedMinusStepIndexDiff	analysis/step_baselines_final.json	paired_vs_baselines.minus_step_index.diff_baseline_minus_model
NumStepFinalPairedMinusStepIndexP	analysis/step_baselines_final.json	paired_vs_baselines.minus_step_index.p_two_sided
NumStepFinalProblems	analysis/step_baselines_final.json	n_problems
NumTargetFinalStepAuc	analysis/target_audit.json	final_step_auc.auc
NumTargetFinalStepAucCiHi	analysis/target_audit.json	final_step_auc.ci95
NumTargetFinalStepAucCiLo	analysis/target_audit.json	final_step_auc.ci95
NumTargetIcc	analysis/target_audit.json	p_hat_variance_components.icc
NumTargetWithinSharePct	analysis/target_audit.json	100 * 1 - p_hat_variance_components.icc
NumTrajAllCompletionsAucMathverse	analysis/parsing_elimination.json	mathverse.all_completions_auc
NumTrajAllCompletionsAucMathvista	analysis/parsing_elimination.json	mathvista.all_completions_auc
NumTrajFullParseOnlyAucMathverse	analysis/parsing_elimination.json	mathverse.full_parse_only_auc
NumTrajFullParseOnlyAucMathvista	analysis/parsing_elimination.json	mathvista.full_parse_only_auc
NumTrajGridTableBody	analysis/selector_summary.json	[*] where scorer in {prm, length}
NumTrajMathverseDiff	analysis/forensics_mathverse.json	paired.diff
NumTrajMathverseDiffCiHi	analysis/forensics_mathverse.json	paired.diff_ci95
NumTrajMathverseDiffCiLo	analysis/forensics_mathverse.json	paired.diff_ci95
NumTrajMathversePairedProblems	analysis/forensics_mathverse.json	paired.n_problems
NumTrajMathvisionDiff	analysis/forensics_mathvision.json	paired.diff
NumTrajMathvisionDiffCiHi	analysis/forensics_mathvision.json	paired.diff_ci95
NumTrajMathvisionDiffCiLo	analysis/forensics_mathvision.json	paired.diff_ci95
NumTrajMathvisionP	analysis/forensics_mathvision.json	paired.p_two_sided
NumTrajMathvisionPairedProblems	analysis/forensics_mathvision.json	paired.n_problems
NumTrajMathvistaDiff	analysis/forensics_mathvista.json	paired.diff
NumTrajMathvistaDiffCiHi	analysis/forensics_mathvista.json	paired.diff_ci95
NumTrajMathvistaDiffCiLo	analysis/forensics_mathvista.json	paired.diff_ci95
NumTrajMathvistaP	analysis/forensics_mathvista.json	paired.p_two_sided
NumTrajMathvistaPairedProblems	analysis/forensics_mathvista.json	paired.n_problems
NumTrajMmmuProDiff	analysis/forensics_mmmu_pro.json	paired.diff
NumTrajMmmuProDiffCiHi	analysis/forensics_mmmu_pro.json	paired.diff_ci95
NumTrajMmmuProDiffCiLo	analysis/forensics_mmmu_pro.json	paired.diff_ci95
NumTrajMmmuProP	analysis/forensics_mmmu_pro.json	paired.p_two_sided
NumTrajMmmuProPairedProblems	analysis/forensics_mmmu_pro.json	paired.n_problems
NumTrajOfflineMathverseAuc	analysis/forensics_mathverse.json	faithful.auc

Table 13 (cont.)

macro	result file	key
NumTrajOfflineMathverseParseRate	analysis/forensics_mathverse.json	faithful.parse_rate
NumTrajOfflineMathvisionAuc	analysis/forensics_mathvision.json	faithful.auc
NumTrajOfflineMathvisionParseRate	analysis/forensics_mathvision.json	faithful.parse_rate
NumTrajOfflineMathvistaAuc	analysis/forensics_mathvista.json	faithful.auc
NumTrajOfflineMathvistaAucN	analysis/forensics_mathvista.json	faithful.n_pairs
NumTrajOfflineMathvistaParseRate	analysis/forensics_mathvista.json	faithful.parse_rate
NumTrajOfflineMathvistaProblems	analysis/forensics_mathvista.json	faithful.n_problems
NumTrajOfflineMmmuProAuc	analysis/forensics_mmmu_pro.json	faithful.auc
NumTrajOfflineMmmuProParseRate	analysis/forensics_mmmu_pro.json	faithful.parse_rate
NumTrajServedMathverseAuc	analysis/forensics_mathverse.json	served.auc
NumTrajServedMathvisionAuc	analysis/forensics_mathvision.json	served.auc
NumTrajServedMathvistaAuc	analysis/forensics_mathvista.json	served.auc
NumTrajServedMathvistaParseRate	analysis/forensics_mathvista.json	served.parse_rate
NumTrajServedMmmuProAuc	analysis/forensics_mmmu_pro.json	served.auc
NumWilsonCoverageTableBody	analysis/label_coverage_M16.json	budgets
NumWpAucMax	analysis/within_problem.json	__summary__.auc_max
NumWpAucMin	analysis/within_problem.json	__summary__.auc_min
NumWpAucMinCell	analysis/within_problem.json	__summary__.auc_min_cell
NumWpBenchmarks	analysis/within_problem.json	__summary__.n_benchmarks
NumWpCells	analysis/within_problem.json	__summary__.n_cells
NumWpCellsScoringFailed	analysis/within_problem.json	__summary__.cells_scoring_failed
NumWpCompletionsCommon	analysis/within_problem.json	__summary__.completions_common
NumWpOursThreeBMathvisionAuc	analysis/within_problem.json	ours_3b_mathvision.auc
NumWpOursThreeBMathvisionAucCiHi	analysis/within_problem.json	ours_3b_mathvision.auc_ci95
NumWpOursThreeBMathvisionAucCiLo	analysis/within_problem.json	ours_3b_mathvision.auc_ci95
NumWpOursThreeBMathvisionGainOverRandom	analysis/within_problem.json	ours_3b_mathvision.gain_over_random
NumWpOursThreeBMathvisionGainOverRandomCiHi	analysis/within_problem.json	ours_3b_mathvision.gain_ci95
NumWpOursThreeBMathvisionGainOverRandomCiLo	analysis/within_problem.json	ours_3b_mathvision.gain_ci95
NumWpOursThreeBMathvisionRandomAcc	analysis/within_problem.json	ours_3b_mathvision.random_acc
NumWpOursThreeBMathvisionTieRatePct	analysis/within_problem.json	100 * ours_3b_mathvision.tie_rate
NumWpOursThreeBMathvisionTopOneAcc	analysis/within_problem.json	ours_3b_mathvision.top1_acc
NumWpParserFailedCells	analysis/within_problem.json	__summary__.parser_failed_cells
NumWpScorers	analysis/within_problem.json	__summary__.n_scorers
NumWpSotaDeepseekROneDistillMmmuProParseRatePct	analysis/within_problem.json	100 * sota_deepseek_r1_distill_mmmu_pro.parse_rate
NumWpSotaQwenTwoFiveVlSevenTwoBInstructMathvistaAuc	analysis/within_problem.json	sota_qwen2_5_vl_72b_instruct_mathvista.auc
NumWpSotaQwenVlThreeBZeroshotMathverseAuc	analysis/within_problem.json	sota_qwen_vl_3b_zeroshot_mathverse.auc
NumWpSotaQwenVlThreeBZeroshotMathverseDistinctScores	analysis/within_problem.json	sota_qwen_vl_3b_zeroshot_mathverse.mean_distinct_scores
NumWpSotaQwenVlThreeBZeroshotMathverseParseRatePct	analysis/within_problem.json	100 * sota_qwen_vl_3b_zeroshot_mathverse.parse_rate
NumWpSotaQwenVlThreeBZeroshotMathvistaParseRatePct	analysis/within_problem.json	100 * sota_qwen_vl_3b_zeroshot_mathvista.parse_rate
NumWpTieRateMaxPct	analysis/within_problem.json	100 * __summary__.tie_rate_max
NumWpTieRateMinPct	analysis/within_problem.json	100 * __summary__.tie_rate_min

H.4 Release contents

Concretely, the release contains: the entry points named in the README (`RELEASE.md` in this repository), a Tier-A/Tier-B split by whether a command needs a GPU; the N=64

completion cache per benchmark under `data/rollouts/` and `data/rollouts_train/`; the step labels with their soft targets under `data/labeled/` and `data/prm_train/`, including the problem-grouped train/test split; every result JSON under `results/analysis/`, plus the per-completion detail files under `results/bon_detail/` that back the forensic reanalysis in App. C; and `MANIFEST.json`, which records a sha256, the paper claim it backs, and the command that regenerates it for every shipped file, machine-checkable against `MANIFEST.sha256`. One command regenerates every table in this paper from that JSON: `python scripts/make_numbers.py --results results --out tex/numbers.tex`, followed by `--check` to fail the build on any macro the sections use that this run did not emit. Two classes of artifact are large and are distributed separately from the JSON tree: the completion caches (multiple gigabytes; shipped only with `collect_release.py --with-rollouts`) and the model weights, the LoRA adapter and the merged bf16 checkpoint, which are not in this tree at all and are published separately with their hashes recorded in `MANIFEST.json`.

H.5 Environment

Every GPU-side measurement in this paper ran on a single machine: 8× NVIDIA B200, Ubuntu 22.04, driver-capped at CUDA 12.8, with torch pinned to the build the driver's CUDA ceiling permits (an unpinned install resolves to a newer CUDA target that imports without error and then reports zero usable devices, which is why the pin is enforced by `scripts/box/00_bootstrap.sh` rather than left to the resolver). The inference stack is vLLM `0.16.0`, serving the merged checkpoint in bf16 with prefix caching and chunked prefill enabled. Training is LoRA on the `Qwen2.5-VL-7B-Instruct` base, rank 256, alpha 512, dropout 0.05, against the soft-target verdict objective (`hard_labels: false`) with the default `k_balanced` sampler, which reweights rows by inverse MC-count-bucket frequency rather than by the binary hard label – a binary class-balanced sampler is explicitly disabled for this run, because it would skew the calibration the soft objective exists to produce (`run_name: prm-vl-7b-lora-r256-balanced`); the one checkpoint on disk for this artifact, `prm_v9_r256_vu_balanced`, is that configuration at one seed – Section 8 states why no seed-repeat or natural-sampler comparison exists. The machine itself is gone: it was ephemeral by construction and was destroyed on 2026-09-23. Nothing about reproducing this paper's numbers depends on that machine specifically, or could – the reproduction path is the released artifact (data, checkpoints, pinned versions, and the JSON this appendix's provenance table traces) together with any machine meeting the same driver-CUDA constraint, not the box the numbers happened to be computed on.