Is the Verifier Worth Its Own Compute? An Equal-FLOP Analysis of Test-Time Verification for Mathematical Reasoning

Anonymous authors
Paper under double-blind review

Abstract

Test-time scaling improves the reasoning accuracy of language models by drawing many samples and selecting one. Two selection strategies dominate: *self-consistency* (majority vote, free) and *verifier-based* selection with a learned reward model (here a Process Reward Model, PRM). The verifier almost always wins *at a fixed number of samples* N – but the verifier is itself a large neural network, and scoring N candidates is not free. We argue that the operative question is not "does the verifier help at equal N?" but "does the verifier help at *equal compute*?" A PRM-scored sample costs the generation FLOPs *plus* the scoring FLOPs; for a 3B generator and a 7B verifier this is $\approx 3.33\times$ the cost of one generation-only sample, so an honest comparison gives self-consistency that many *more* samples. We formalize a coverage/realized/verifier-efficiency decomposition with unbiased estimators and bootstrap confidence intervals, and an equal-FLOP comparison on a common compute axis. On GSM8K the verifier earns its compute from a small budget ($k\approx4$); MATH-500 (N=64) is harder and more revealing. Across *five* generators spanning four families (Qwen, Phi, Llama, Mistral, gemma; the 7B+ models in 4-bit), the verifier's compute-worth follows a simple, measurable rule – its per-sample selection advantage over majority vote, Δ, weighed against its cost ratio r: the verifier is worth its compute only once Δ is large enough to beat r. For competent generators Δ is modest, so at small budgets self-consistency *significantly* beats the verifier – when compute is scarce, more samples win – and the crossover, where it occurs at all, ranges from k=8 to beyond a practical budget. The same rule accounts for the exception: Mistral-7B's self-consistency is nearly useless, so its verifier wins from k=2. Δ is decisive, not the tax: gemma has the lowest r yet its verifier never wins (tiny Δ), while Mistral wins early (real Δ). The finding is robust across four PRM step-aggregations and a second, different-type (outcome) reward model.

1 Introduction

Scaling *inference* compute – sampling many candidate solutions and selecting among them – is now a standard lever for improving the reasoning accuracy of large language models (Brown et al., 2024; Snell et al., 2024). Given a base model, two selection rules account for most practical gains. **Self-consistency** (Wang et al., 2023) takes a plurality vote over the answers of N sampled solutions and costs nothing beyond generation. **Verifier-based selection** (Cobbe et al., 2021; Lightman et al., 2024) scores each candidate with a learned reward model – increasingly a *Process Reward Model* (PRM) that scores reasoning steps – and returns the best-scoring one.

A large literature reports that verifiers beat majority voting *at a fixed sample budget* N. But this comparison silently ignores the verifier's own cost. A PRM is a full transformer (often as large as, or larger than, the generator); scoring N candidates requires N additional forward passes. When the budget is *compute* rather than *sample count*, the verifier must justify not only its selections but the FLOPs those selections consume – FLOPs that self-consistency could instead spend generating and voting over more samples.

This paper. We ask: *is the verifier worth its own compute?* We make the comparison fair by placing both strategies on a single compute axis measured in generation-sample-equivalents. Concretely, if generating one sample costs P_{gen} and scoring it costs P_{score} (in parameters, hence FLOPs), then one verifier-selected sample costs $r = (P_{\text{gen}} + P_{\text{score}})/P_{\text{gen}}$ as much as one self-consistency sample, and we compare PRM best-of-k against self-consistency best-of-$(r\,k)$. Our contributions:

1. **A decomposition with unbiased estimators.** We separate the achievable ceiling (*coverage*, =pass@k), what a real selector achieves (*realized* accuracy), and the fraction of available headroom it captures (*verifier efficiency*). Realized accuracy is defined as an expectation over random k-subsets with uniform tie-breaking, which makes realized(1) ≡pass@1 and removes a subtle small-k bias. All quantities carry bootstrap confidence intervals over problems.

2. **An equal-FLOP comparison** that asks who wins at each compute budget, with explicit, stated FLOP accounting and a cost ratio r that depends on the generator/verifier size pair.

3. **A measured mechanism for verifier value, across five generators and four families** (Qwen2.5-3B, Phi-3.5-mini, Llama-3.2-3B, Mistral-7B, gemma-2-9B; verifier Qwen2.5-Math-PRM-7B): whether the verifier beats self-consistency at equal FLOPs is governed by its selection advantage over majority vote, Δ, weighed against its cost ratio r – the verifier wins once Δ beats r (§5.5). This yields the common case – for competent generators Δ is modest, so at small budgets self-consistency *significantly* wins (more samples beat a verifier) – and accounts for its exception: a very weak generator (Mistral), whose self-consistency is useless, has a large Δ and the verifier wins from k=2. The gemma-vs-Mistral contrast (both low r, opposite outcomes) shows Δ, not the tax, is decisive. The finding is robust across four PRM aggregations and a second, different-type (outcome) verifier (§5.6).

We frame one thing as preliminary: the study uses one verifier family and 7B+ generators only in 4-bit (larger full-precision runs are future work, §7). The crossovers are *measured* – N=64 places the matched self-consistency budget $(r\,k)$ inside the data across the range.

2 Related Work

Repeated sampling and coverage. Drawing many samples raises the chance that *some* sample is correct; pass@k (coverage) grows smoothly with k (Chen et al., 2021; Brown et al., 2024). Coverage is an oracle upper bound: it assumes a perfect selector. The gap between coverage and what a real selector achieves is exactly what verifiers try to close.

Selection: voting vs. verifiers. Self-consistency (Wang et al., 2023) is the standard free baseline. Outcome- and process-reward verifiers (Cobbe et al., 2021; Lightman et al., 2024) improve selection and are central to recent math models (Qwen Team, 2024). PRMs score intermediate steps and are typically aggregated (product/min/last/mean) into a sequence score.

Compute-optimal inference. Snell et al. (2024) and Wu et al. (2024) study how to allocate a test-time budget, and observe that the best strategy depends on difficulty. Our work sharpens one axis of this question into a controlled, like-for-like equal-FLOP comparison that *charges the verifier for its own forward passes*, and reports the decomposition with confidence intervals so the reader can see whether differences exceed noise. To our knowledge the specific question "does charging the verifier its own FLOPs erase its apparent advantage, and does the answer depend on task difficulty?" has not been reported with this accounting.

3 Method

3.1 Decomposition

For a problem with N sampled solutions, let c be the number that are correct. **Coverage** at budget k is the probability that a random k-subset contains at least one correct solution – the standard unbiased pass@k

estimator (Chen et al., 2021):

$$\text{Coverage}(k) \;=\; \mathbb{E}[\text{pass@}k] \;=\; 1 - \frac{\binom{N-c}{k}}{\binom{N}{k}}. \tag{1}$$

Realized accuracy of a selection strategy s at budget k is the probability that s, applied to a random k-subset, returns a correct answer. We compute it as an expectation over k-subsets with *uniform tie-breaking* (exact enumeration when $\binom{N}{k}$ is small, Monte-Carlo otherwise). This definition guarantees Realized(1) = pass@1 and avoids a downward bias at small k that arises if one instead scores a single fixed ordering. **Verifier efficiency** is the fraction of the available headroom that the strategy captures:

$$\text{VerifEff}(k) \;=\; \frac{\text{Realized}(k) - \text{pass@1}}{\text{Coverage}(k) - \text{pass@1}}, \tag{2}$$

defined when Coverage(k) > pass@1 (else reported as N/A). VerifEff = 1 means the selector is as good as an oracle; 0 means it does no better than a single sample. All quantities are averaged over problems, with 95% bootstrap confidence intervals computed by resampling problems.

3.2 Cost accounting

We use the standard dense-transformer approximation: a forward pass over T tokens of a P-parameter model costs $\approx 2PT$ FLOPs (Kaplan et al., 2020; Hoffmann et al., 2022). Generating a sample with prompt length T_{in} and output length T_{out} costs $\approx 2P_{\text{gen}}(T_{\text{in}} + T_{\text{out}})$; scoring a candidate of length L with a verifier of P_{ver} parameters costs $\approx 2P_{\text{ver}}(T_{\text{in}} + L)$ (one forward pass, no autoregressive decode). We report token counts alongside FLOPs so readers can re-derive costs under their own model.

3.3 Equal-FLOP comparison

One verifier-selected sample (generate + score) costs

$$r \;=\; \frac{P_{\text{gen}} + P_{\text{ver}}}{P_{\text{gen}}} \tag{3}$$

times one generation-only (self-consistency) sample. For our 3B generator and 7B verifier, $r = (3 + 7)/3 \approx 3.33$. To match compute, we compare PRM best-of-k against self-consistency best-of-$(r\,k)$, interpolating the self-consistency curve in $\log_2$ budget. We then read off, at each budget, which strategy wins and by how much, flagging any point whose matched self-consistency budget exceeds N as *extrapolated* (i.e. beyond measured data). Note that r *shrinks as the generator grows*: a 7B generator with the same 7B verifier gives $r = 2.0$, so larger generators make the verifier easier to justify – an axis we exploit in the multi-architecture extension.

4 Experimental Setup

Generator. Qwen2.5-3B-Instruct, sampled at temperature 0.8, top-p 0.95. **Verifier.** Qwen2.5-Math-PRM-7B (Qwen Team, 2024); we split each solution into steps on blank lines, score steps at the model's `<extra_0>` positions, and aggregate by `min` (most robust in our runs; product/last/mean also stored). Our PRM integration reproduces the model card's reference step rewards to within numerical tolerance before any scoring run. **Datasets.** GSM8K (Cobbe et al., 2021) (200 test problems, $N = 64$, max 512 new tokens) and MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024) (all 500 problems labeled by difficulty level 1–5, $N = 64$, max 1024 new tokens). **Estimators.** Coverage via the combinatorial pass@k formula; realized accuracy by exact k-subset enumeration (or 2000-sample Monte-Carlo when $\binom{N}{k}$ is large); 95% CIs by problem bootstrap (2000 resamples). **Reproducibility.** Fixed seeds; full generated text stored so answer extraction can be revised and re-scored without re-generating. Generation used vLLM (with its native, non-FlashInfer sampler) under WSL2 on a single 16 GB consumer GPU; the in-progress multi-architecture sweep uses the same stack.

Table 1: GSM8K decomposition (Qwen2.5-3B-Instruct, N=64, 200 problems). Values are mean [95% CI]. "SC" = self-consistency (majority vote), "PRM" = best-of-k by Qwen2.5-Math-PRM-7B (`min` aggregation).

k	Coverage	Realized(SC)	Realized(PRM)	VerifEff(SC)	VerifEff(PRM)
1	0.813	0.813	0.813	—	—
2	0.882 [.843,.915]	0.813	0.870 [.830,.904]	0.00	0.83
4	0.920 [.885,.949]	0.865 [.822,.903]	0.903 [.866,.935]	0.49	0.84
8	0.944 [.913,.968]	0.886 [.844,.923]	0.922 [.886,.952]	0.56	0.84
16	0.959 [.932,.981]	0.897 [.856,.934]	0.933 [.899,.961]	0.58	0.82
64	0.975 [.950,.995]	0.913 [.870,.950]	0.937 [.902,.967]	0.62	0.76

Table 2: equal-FLOP comparison on GSM8K (r=3.33). PRM best-of-k vs. self-consistency at the equal-FLOP budget $r\,k$. Positive margin = PRM wins. **Bold** = 95% paired-bootstrap CI excludes zero.

PRM k	PRM acc	equal-FLOP SC budget	SC acc	margin (PRM−SC)
1	0.813	3.3	0.851	−0.039
2	0.870	6.7	0.880	−0.011
4	0.903	13.3	0.894	+0.009
8	0.922	26.7	0.903	+0.019
16	0.933	53.3	0.911	+0.023

5 Results

5.1 GSM8K: the verifier earns its compute

Table 1 reports the decomposition. The base model already solves 81.3% of problems (pass@1); coverage rises to 97.5% at k=64, so most of the headroom is reachable in principle. Self-consistency captures roughly 60% of that headroom; the PRM captures ∼80% at every budget. Extraction failure was 0%.

The equal-FLOP verdict (Table 2) is the headline for GSM8K, and it already shows the budget-threshold shape. At small budgets self-consistency *significantly* wins (the verifier tax is not yet amortized); the strategies cross near k=4; and from k=8 the PRM is *significantly* ahead (+0.019, CI $[+0.005, +0.038]$ at k=8; +0.023, CI $[+0.003, +0.045]$ at k=16). This is *measured* (matched self-consistency budget $\sim 53 \leq 64$). On this easy task the threshold is low: the verifier is worth its compute from a small budget.

5.2 MATH-500: headroom and selection vs. difficulty

Table 3 stratifies the decomposition by MATH difficulty level. Three patterns are clear and monotone in difficulty: (i) pass@1 falls from 0.87 (L1) to 0.33 (L5); (ii) coverage also falls (the L5 ceiling is only 0.78 even at k=64, so on the hardest problems the base model often cannot produce a correct solution at all); and (iii) at *equal* N, the PRM is the better selector at every level, capturing more of the available headroom than self-consistency (overall verifier efficiency 0.60 vs. 0.46; L5 realized $0.511 \rightarrow 0.555$). This equal-N advantage is precisely what the equal-FLOP accounting in §5.3 puts to the test. Extraction failure on MATH was 10.8% (vs. 0% on GSM8K); notably, *every* such failure is a generation truncated at the 1024-token cap (no answer was ever emitted), concentrated on the hardest problems (L5: 20%), rather than a parser miss – so counting them wrong is correct, and the effect falls identically on both the self-consistency and verifier arms (see §7).

5.3 MATH-500: is the verifier worth its compute?

Under equal-FLOP accounting the equal-N advantage does not vanish – it acquires a *budget threshold* (Table 4). At small budgets the verifier's 3.33× tax is not amortized and self-consistency wins *significantly*: overall, the margin is −0.052 at k=1 (CI $[-0.058, -0.045]$) and −0.028 at k=2 (CI $[-0.036, -0.021]$), both excluding zero. The strategies cross at $k \approx 4$–8, and by k=16 the PRM is ahead by +0.022 with a CI that also

Table 3: MATH-500 decomposition by difficulty level (Qwen2.5-3B-Instruct, N=64, k=64, PRM `min`). n is problems per level; VE = verifier efficiency (fraction of headroom captured). All values are means; see text for CIs.

Level	n	pass@1	Coverage	Realized(SC)	Realized(PRM)	VE(SC)	VE(PRM)
L1	43	0.873	0.953	0.907	0.953	0.42	1.00
L2	90	0.805	0.956	0.867	0.911	0.41	0.70
L3	105	0.730	0.924	0.848	0.886	0.61	0.80
L4	128	0.549	0.867	0.705	0.730	0.49	0.57
L5	134	0.327	0.784	0.511	0.555	0.40	0.50
All	500	0.602	0.880 [.852,.910]	0.730 [.690,.767]	0.767 [.732,.804]	0.46	0.60

Table 4: equal-FLOP comparison on MATH-500 (r=3.33), overall and at level 5, all rows *measured* (N=64: the matched SC budget $r\,k \leq 53.3 \leq 64$). Margin = PRM−SC; positive = PRM wins. **Bold** = 95% paired-bootstrap CI excludes zero.

All levels (pass@1 = 0.602)				**Level 5** (pass@1 = 0.327)			
PRM k	PRM	SC(rk)	margin	PRM k	PRM	SC(rk)	margin
1	0.602	0.653	−0.052	1	0.327	0.385	−0.058
2	0.659	0.688	−0.028	2	0.397	0.429	−0.032
4	0.700	0.708	−0.007	4	0.454	0.459	−0.004
8	0.729	0.720	+0.010	8	0.497	0.479	+0.018
16	0.749	0.728	+0.022	16	0.524	0.504	+0.020

excludes zero ($[+0.004, +0.041]$) – a *measured* advantage, since the matched self-consistency budget $r\,k \approx 53$ lies within N=64. The same shape holds at every level, with the crossover shifted to larger k on harder problems (L5 crosses near k=8); per level the k=16 verifier margins are positive but, at 43–134 problems each, not individually significant. Margins are computed by a *paired* problem bootstrap (the same resampled problems score both arms), and their point values reproduce the interpolated PRM − SC differences exactly. The reading: *the verifier is worth its compute on MATH too, but only above a budget threshold ($k \approx 8$–16) higher than GSM8K's; below it, spending the verifier's FLOPs on more samples is the better – and significantly better – choice.* This is the picture for one generator; §5.4 shows how it varies across five, and §5.5 identifies the single quantity that sets where the crossover lands.

5.4 Generalization across architectures

Table 5 repeats the MATH-500 equal-FLOP analysis across *five* generators spanning *four families* (Qwen, Phi, Llama, Mistral, gemma), with the same verifier; the 7B+ generators run in 4-bit AWQ to fit our 16 GB GPU. The crossover budget varies enormously – from k=2 to beyond the measured range – and the value of this sweep is that it lets us identify *what governs* that variation (§5.5).

For four of the five generators, self-consistency *significantly* beats the verifier at small budgets: when the budget is small, spending the verifier's FLOPs on more samples is the compute-optimal choice. The verifier's later overtaking is generator-dependent – measured for Qwen (k=16) and Phi (k=8), but never in range for Llama and gemma. The lone exception to the small-budget rule is **Mistral-7B**, a very weak generator (pass@1 0.11) whose self-consistency is nearly useless; there the verifier wins from k=2. Rather than a caveat, this exception is *explained* by the mechanism below: it is not that Mistral breaks the pattern, but that the same quantity governing every generator's crossover simply lands early for Mistral.

5.5 What drives the crossover

The crossover budget varies widely across generators (Table 5) – from k=2 (Mistral) to never-in-range (Llama, gemma). We identify what governs it. Using verifier efficiency – the fraction of available headroom (Coverage−pass@1) a selector captures – the key quantity is the PRM's efficiency advantage *over majority*

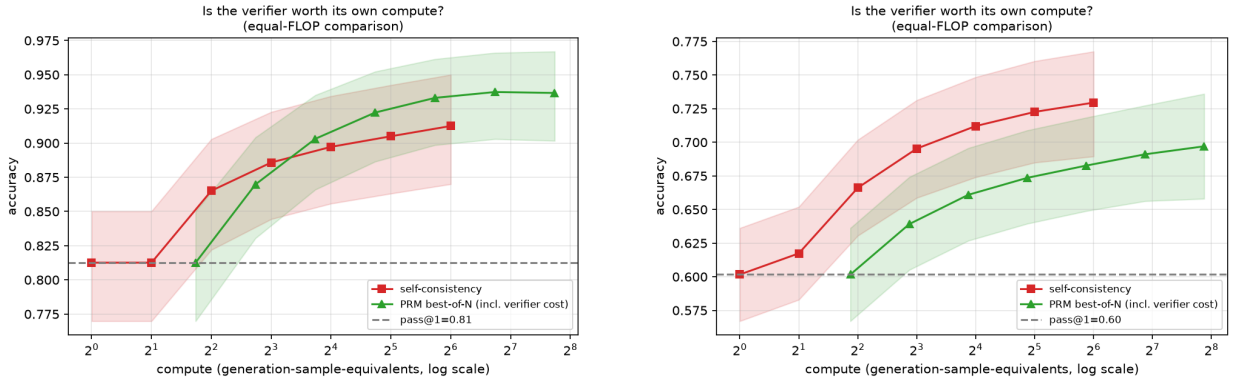

Figure 1: Accuracy vs. compute (generation-sample-equivalents, log scale) for GSM8K (left) and MATH-500 (right). The PRM curve includes its own verifier cost (r=3.33). For this generator the verifier overtakes equal-FLOP self-consistency at a small budget on GSM8K and a larger one on MATH; whether that difficulty gap holds across generators is examined in §5.4.

Table 5: Budget threshold across five generators (four families) on MATH-500 (N=64; verifier Qwen2.5-Math-PRM-7B, `min`). "SC/PRM sig. wins" give budgets at which the 95% paired-bootstrap margin excludes zero, within the measured range ($r\,k \leq N$). Self-consistency wins at small budgets on four of five; the exception is Mistral-7B (a very weak generator), where the verifier wins from k=2 – explained by the mechanism in §5.5.

Generator	family	params	r	pass@1	Cov@64	SC wins	PRM wins
Qwen2.5-3B-Instruct	Qwen	3.0B	3.33	0.602	0.880	$k \leq 2$	$k \geq 16$
Phi-3.5-mini-instruct	Phi	3.8B	2.84	0.392	0.810	$k \leq 2$	$k \geq 8$
Llama-3.2-3B-Instruct	Llama	3.0B	3.33	0.369	0.844	$k \leq 8$	none†
Mistral-7B-Instruct (AWQ)	Mistral	7.2B	1.97	0.108	0.592	$k \leq 1$	$k \geq 2$
gemma-2-9b-it (AWQ)	gemma	9.2B	1.76	0.423	0.756	$k \leq 8$	none†

†PRM does not significantly win at any measured budget (crossover extrapolated to $k \gtrsim 32$). The 7B+ generators use 4-bit AWQ weights; the verifier is bf16 for all generators.

vote, $\Delta = \mathrm{VE(PRM)} - \mathrm{VE(SC)}$ (Table 6). The verifier is worth its FLOP-tax only when Δ is large enough to overcome the tax: it must out-select the free baseline by enough to pay for the r extra samples self-consistency could have drawn instead.

Two contrasts isolate the drivers. **Δ is decisive.** Where Δ is tiny the verifier never wins, *regardless of tax*: gemma has the *lowest* cost ratio in the set (r=1.76) yet its verifier never crosses, because its self-consistency is already efficient (Δ=0.04, like Llama). Where Δ is large the verifier wins early: Mistral's self-consistency is nearly useless (Δ=0.17) so the verifier crosses at k=2. **The tax r shifts *where*.** Qwen and Mistral have comparable Δ (0.14 vs 0.17) but Qwen's higher tax (r=3.33 vs 1.97) pushes its crossover out to k=16 against Mistral's k=2. So the crossover follows a two-factor rule – the verifier wins once its selection advantage Δ beats its tax r – with Δ the necessary ingredient: a cheap verifier cannot rescue a small Δ (gemma), while a real Δ pays off quickly when the tax is low (Mistral). The gemma-vs-Mistral contrast (both low-r, opposite outcomes) shows the driver is *selection advantage*, not the tax.

What sets Δ: answer-distribution entropy. Because Δ is large exactly when self-consistency is a weak selector, we can ask *why* voting is weak. Across the five generators, VE(SC) is strongly predicted by the entropy of the model's answer distribution (Pearson $r = -0.91$; Fig. 3): generators whose samples concentrate on few answers vote well (Qwen, entropy 0.24, VE(SC) 0.46), while diffuse ones vote poorly (Mistral, entropy 0.78, VE(SC) 0.25). Splitting the answer mass by correctness locates the cause in *quiet correct answers*, not confident wrong ones: VE(SC) tracks how loudly the correct answer is voted ($r = +0.84$)

Table 6: What drives the crossover: the PRM's verifier-efficiency advantage over self-consistency, Δ = VE(PRM) − VE(SC), and the cost ratio r (MATH-500, N=64, k=64, `min`). The verifier wins once Δ is large enough to beat r. Δ is decisive: gemma has the lowest r yet never wins (tiny Δ), while Mistral wins at k=2 (real Δ, low r) – so selection advantage, not the tax, governs.

Generator	VE(SC)	VE(PRM)	Δ	r	crossover
Mistral-7B (AWQ)	0.25	0.42	+0.17	1.97	$k \approx 2$
Phi-3.5-mini	0.36	0.55	+0.19	2.84	$k \approx 8$
Qwen2.5-3B	0.46	0.60	+0.14	3.33	$k \approx 16$
gemma-2-9b (AWQ)	0.39	0.43	+0.04	1.76	none
Llama-3.2-3B	0.45	0.49	+0.04	3.33	none

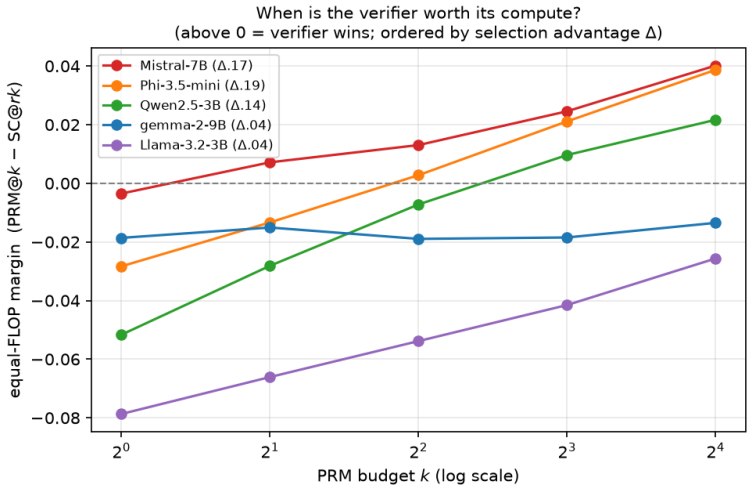

Figure 2: Equal-FLOP margin (verifier − self-consistency) vs. budget for all five generators. The verifier is worth its compute where a curve rises above zero; the crossover is ordered by the selection advantage Δ (larger Δ crosses earlier), with the cost ratio r shifting it. Mistral crosses at k=2, Phi near k=8, Qwen near k=16; gemma and Llama stay below zero throughout the measured range.

but is nearly independent of the loudest wrong answer ($r = +0.36$). Mistral is not confidently wrong – it is diffuse everywhere, its correct answer whispering at a 0.18 vote share against a 0.12 top distractor. The verifier's role is thus to *re-concentrate* mass that voting scatters: it pays off when the correct answer is present (coverage) but not loud enough to win a vote. With five generators these correlations are directional, not significance-tested (§7), but they turn "what sets Δ" from a black box into a measurable property of the generation distribution.

5.6 Robustness: aggregation and a second verifier

Two checks probe whether the small-budget dominance of self-consistency is an artifact of our verifier choices. **(i) PRM aggregation.** Repeating the analysis under all four step-score aggregations (`min`/`last`/`mean`/`product`) leaves the small-budget result intact: self-consistency significantly wins at $k \leq 2$ (Qwen, Phi) or through $k \leq 8$ (Llama) under *every* aggregation. The verifier's *overtaking*, by contrast, is aggregation-fragile – for Qwen the PRM wins under `min` and `product` but *not* under `last` or `mean`. So the aggregation-invariant fact is precisely that self-consistency is compute-optimal at small budgets. **(ii) A second, independent verifier.** We repeat the equal-FLOP comparison with an *outcome* reward model of a different family (Skywork-Reward-Llama-3.1-8B, 4-bit; $r = (P_{\text{gen}}+8)/P_{\text{gen}}$) in place of the process PRM, on the three ∼3B-tier generators. Under this verifier, self-consistency *significantly wins at every measured budget for all three* (Table 7) – the outcome RM never justifies its compute. It is a weaker selector than the

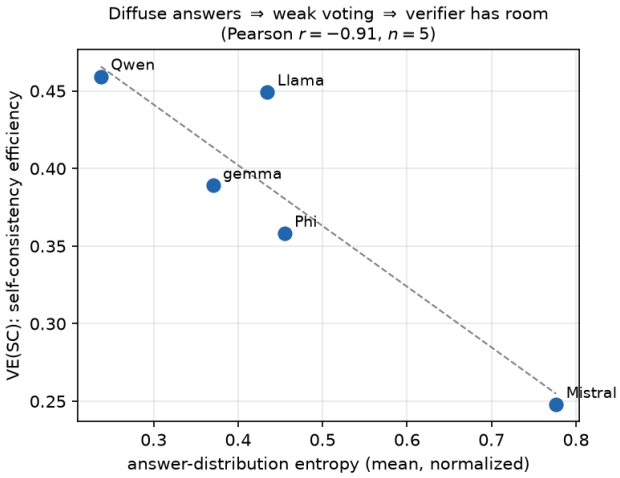

Figure 3: Why self-consistency is a strong or weak selector: VE(SC) vs. the entropy of the generator's answer distribution. Diffuse (high-entropy) outputs make majority vote a poor selector, leaving headroom the verifier can capture (large Δ); peaked outputs already vote well. Pearson $r = -0.91$; directional at n=5.

Table 7: Second verifier – an outcome RM (Skywork-Reward-Llama-3.1-8B, 4-bit) – on MATH-500 (N=64), replacing the process PRM. Budgets where the 95% paired-bootstrap margin (verifier−SC) excludes zero. Self-consistency significantly wins at *every* measured budget for all three ∼3B-tier generators; the outcome RM never wins.

Generator	r	SC sig. wins	ORM sig. wins
Qwen2.5-3B-Instruct	3.67	all ($k \leq 16$)	none
Phi-3.5-mini-instruct	3.11	all ($k \leq 16$)	none
Llama-3.2-3B-Instruct	3.67	all ($k \leq 16$)	none

process PRM (it is general-purpose, not math-specialized), so we do not read this as "verifiers are useless"; rather, the small-budget self-consistency advantage is robust across verifier *type* (process and outcome), and beating self-consistency at equal FLOPs requires a strong, specialized verifier *and* a large budget.

5.7 Summary of findings

- **GSM8K (easy):** verifier crosses over near $k \approx 4$ and is *significantly* ahead from k=8 (measured).
- **MATH-500 (hard), Qwen2.5-3B:** a budget threshold – self-consistency *significantly* wins at $k \leq 2$; the verifier overtakes at $k \approx 8$ and is significantly ahead by k=16 (pooled, measured, +0.022 with CI $[+0.004, +0.041]$).
- **Mechanism, across five generators (four families):** whether the verifier beats self-consistency at equal FLOPs is set by its selection advantage Δ over majority vote versus its cost ratio r; crossovers span k=2 (Mistral) to never (Llama, gemma). Δ is decisive – gemma has the lowest r yet never wins (tiny Δ), while Mistral wins early (real Δ); r only shifts where. For competent generators Δ is modest, so self-consistency significantly wins at small budgets (§5.5).
- **Robustness (§5.6):** the small-budget self-consistency win holds under all four PRM aggregations; under a second, different-type verifier (an outcome RM), self-consistency significantly wins at *every* budget for all generators. Beating self-consistency at equal FLOPs is the contingent case – it needs a real Δ (and, for costly verifiers, a large budget).

6 Discussion

The standard equal-N comparison flatters verifiers because it does not charge them for the compute they consume. Once charged, whether a verifier is worth it is governed by a two-factor rule (§5.5): its per-sample selection advantage over majority vote, Δ, versus its cost ratio r. The verifier wins only once Δ is large enough to beat r, and this organizes all five generators.

Δ is the decisive factor. Where Δ is small the verifier never wins, regardless of tax – gemma, with the *lowest* r in our set, never crosses because its self-consistency is already an efficient selector; Llama is the same. Where Δ is large the verifier wins even at small budgets – Mistral, whose self-consistency is nearly useless, crosses at k=2. The cost ratio r only modulates where a given Δ crosses: Qwen and Mistral have comparable Δ, but Qwen's higher tax pushes its crossover from k=2 out to k=16. The practical upshot: for competent generators, at small budgets sample more and vote; a verifier earns its compute only when it out-selects the free baseline by a wide margin, or when the base model is so weak that voting is hopeless.

What sets Δ itself – how good self-consistency already is on a generator's outputs (its answer diversity and consensus) – we do not fully resolve. It tracks task difficulty loosely (within Qwen, harder levels raise the crossover, and GSM8K < MATH) but not monotonically across generators. Predicting Δ from a generator's generation statistics alone is a natural next step.

7 Limitations and Ongoing Work

We state limitations prominently because they bound the current claims.

1. **Model scope; quantized large generators.** Results span five generators across four families (Qwen, Phi, Llama, Mistral, gemma) and two verifiers of different type and family (a process PRM, Qwen2.5-Math-PRM-7B; an outcome RM, Skywork-Reward-Llama-3.1-8B). The 7B+ generators run in 4-bit AWQ to fit a 16 GB GPU (the verifier is bf16 for all); full-precision large generators are unverified.

2. **What sets Δ is only partly resolved.** We show Δ governs the crossover and that VE(SC) tracks the entropy of the generator's answer distribution (§5.5, Fig. 3, $r = -0.91$) – but with five generators this correlation is directional, not significance-tested. A larger generator panel is needed to establish it as a predictive law.

3. **Generation budget.** We cap generation at 1024 tokens. On MATH this truncates 10.8% of solutions (rising to 20% at L5) before a final answer is emitted; we count these wrong, which is correct (no answer was produced) but lowers absolute accuracies on the hardest problems. Crucially the cap falls identically on the self-consistency and verifier arms, so it does not bias the equal-FLOP comparison; a larger budget would raise the ceilings, most at L5.

4. **Statistical power.** The low-budget self-consistency wins and the pooled k=16 MATH verifier win are significant, but several per-level margins are within bootstrap CIs; the multi-architecture runs and additional problems will tighten them.

5. **Answer extraction is not a source of error.** Every MATH extraction failure is a truncated generation (item above), not a parser miss, and gold-answer extraction never fails; scoring validity is therefore not limited by the answer parser. Full generations are stored so any future re-scoring needs no re-generation.

6. **Verifier coverage.** We test two verifiers – a math process PRM and a general-purpose outcome RM – under four aggregations (§5.6). Untested are a *math-specialized* outcome model and verifiers of other sizes (varying r at a fixed generator would isolate the cost-tax effect from generator capability).

8 Conclusion

Charging a test-time verifier for its own forward passes changes the picture: a learned verifier that clearly wins at equal sample count need not win at equal *compute*. Across five generators and four families, whether it does is governed by a simple rule – its per-sample selection advantage over majority vote, Δ, must beat its

cost ratio r. In the common case (Δ modest) self-consistency significantly wins at small budgets, so scarce compute is better spent sampling more; the *same* rule accounts for the exception – a very weak generator, whose voting is hopeless, where the verifier wins from k=2. Δ, not the tax, is decisive: the cheapest verifier still loses when its selection advantage is small. The result is robust across four PRM aggregations and a second, different-type verifier. We release the decomposition, unbiased estimators, and equal-FLOP methodology; characterizing what makes self-consistency a strong or weak selector – i.e. what sets Δ – is the natural next step.

Reproducibility Statement

We report unbiased estimators with bootstrap CIs, fixed seeds, exact sampling hyperparameters, and store all generated text so extraction and scoring are reproducible offline. The cost model is the explicit $2PT$ approximation with token counts reported alongside FLOPs. Code for generation, PRM scoring, the decomposition, and the equal-FLOP analysis accompanies the paper.

References

B. Brown, J. Juravsky, et al. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. *arXiv:2407.21787*, 2024.

M. Chen, J. Tworek, et al. Evaluating Large Language Models Trained on Code. *arXiv:2107.03374*, 2021. (pass@k estimator.)

K. Cobbe, V. Kosaraju, et al. Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168*, 2021. (GSM8K.)

D. Hendrycks, C. Burns, et al. Measuring Mathematical Problem Solving With the MATH Dataset. *NeurIPS Datasets and Benchmarks*, 2021.

J. Hoffmann, S. Borgeaud, et al. Training Compute-Optimal Large Language Models. *arXiv:2203.15556*, 2022.

J. Kaplan, S. McCandlish, et al. Scaling Laws for Neural Language Models. *arXiv:2001.08361*, 2020.

H. Lightman, V. Kosaraju, et al. Let's Verify Step by Step. *ICLR*, 2024. (PRM800K; MATH-500 subset.)

C. Snell, J. Lee, K. Xu, A. Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv:2408.03314*, 2024.

X. Wang, J. Wei, et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ICLR*, 2023.

Y. Wu, Z. Sun, et al. An Empirical Analysis of Compute-Optimal Inference for Problem-Solving with Language Models. *arXiv:2408.00724*, 2024.

Qwen Team. Qwen2.5-Math Technical Report (Qwen2.5-Math-PRM-7B). *arXiv:2409.12122*, 2024.