KNOW WHAT YOU'VE SELECTED: ON-POLICY RECALIBRATION FOR BEAM-SEARCH-INDUCED PRM DRIFT

Anonymous authors
Paper under double-blind review

ABSTRACT

Process reward models (PRMs) guide reasoning search by scoring intermediate steps, offering signal on solution quality before completion—particularly useful for weaker models on complex tasks, where methods like Best-of-N and self-consistency only evaluate complete solutions. Standard beam search is expensive, generating $K \times N$ candidates per depth, so recent work uses calibrated PRMs to adaptively shrink search width. We show this assumption fails: PRM calibration is typically done on prefixes sampled independently of search, but deployment evaluates prefixes that survived search-induced pruning at every prior step. This creates a self-referential covariate shift that compounds with depth, causing PRMs to grow increasingly overconfident on deeper, survivorship-selected prefixes—even after calibration—across both Qwen-PRM and Math-Shepherd. We propose an on-policy correction that recalibrates PRMs using their own search-induced prefixes, substantially reducing this depth-dependent overestimation for both PRMs. Downstream, this translates into consistent beam-search accuracy gains with Qwen-PRM across generators and benchmarks, with gains concentrated at moderate self-consistency uncertainty (up to 10% relative to extremes) and additional search budget allocated preferentially to lower-certainty questions; gains with Math-Shepherd are more mixed, improving in most but not all configurations. These results suggest the covariate-shift diagnosis is general, while its downstream accuracy impact depends on the underlying PRM.

1 INTRODUCTION

Process reward models (PRMs) (Lightman et al., 2024; Zhang et al., 2025) act as external verifiers for models with weak reasoning capabilities, and have been shown to be effective at ranking intermediate steps and reasoning paths in test-time methods such as Best-of-N.

For a fixed sampling budget, this ranking capability alone is sufficient to improve accuracy, at the cost of the extra compute spent evaluating every candidate (Park et al., 2025b). Since many questions do not require this full budget, adaptive sampling techniques instead allocate less compute to easier questions and more to harder ones, saving compute without sacrificing accuracy (Aggarwal et al., 2023; Taubenfeld et al., 2025). However, for sufficiently complex reasoning tasks, sampling-based techniques—self-consistency, outcome reward models, and PRM-based aggregation over complete solutions—may require more samples than any adaptive budget can afford, since none of these methods can intervene before a full candidate solution has already been generated. Search-based methods such as beam search (Xie et al., 2023) and Monte Carlo Tree Search (MCTS) (Chen et al., 2024; Hao et al., 2023) address this by refining intermediate steps at every depth, retaining and controlling solution quality throughout generation rather than only at the end.

This benefit comes at a cost: these methods depend on a calibrated verifier to determine how much budget each step actually needs, and PRMs are known to be miscalibrated in a specific, consequential way—they tend to overestimate step-level scores, causing search algorithms to under-allocate budget precisely where more budget is needed. Park et al. (2025b) address this by explicitly calibrating PRMs to penalize overestimation, substantially reducing compute cost on datasets similar to their

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.

calibration distribution (e.g., MATH), though this benefit is markedly weaker on out-of-distribution benchmarks such as AIME.

This calibration procedure, however, inherits a structural assumption from how PRM training data is collected: full reasoning paths are sampled independently, then pruned and labeled after the fact to produce step-level correctness scores. Every prefix in this training data is therefore drawn *unconditionally* from the policy's own generation distribution. We identify a shortcoming in this assumption when the calibrated PRM is deployed directly within beam search. In beam search, whether a step at depth d is even evaluated depends not only on the autoregressive likelihood of generating that step, but also on whether every preceding step from depth 0 to $d-1$ survived PRM-based pruning. In the PRM's training data, by contrast, a step at depth d reflects only its autoregressive generation likelihood, with no such survivorship conditioning. This mismatch means the PRM is evaluated at deployment on a progressively different distribution than the one it was calibrated on—one that the PRM itself has shaped through its own pruning decisions at every preceding step.

We show, both formally and empirically, that this self-referential covariate shift causes PRM scores to become increasingly overestimated as search depth increases, even after explicit calibration for overestimation, and that this phenomenon holds across two PRMs with different training methodologies—Qwen-PRM and Math-Shepherd. We address this with an on-policy correction that recalibrates the PRM on prefixes drawn from its own search-induced distribution, substantially reducing this depth-dependent overestimation. Using the recalibrated Qwen-PRM checkpoint, we show downstream beam-search accuracy improves, particularly for generators whose questions concentrate in a region of partial certainty. Stratifying by self-consistency entropy, we find this gain is not uniform across question difficulty: it is largest at moderate uncertainty, reaching up to 10 percentage points over the accuracy gain observed at the extremes. This improvement is not simply a matter of spending more compute—the recalibrated PRM allocates additional search budget preferentially to questions of partial and low certainty, rather than uniformly across all questions.

Our contributions are as follows:

- We diagnose a covariate shift that occurs when using PRMs for beam search, which grows with search depth. We observe that this covariate shift causes calibrated PRMs to retain the overestimation problem they were explicitly trained to avoid. We propose an on-policy recalibration technique that reduces this overestimation and improves downstream beam-search accuracy on a weak reasoner, Llama-3.2-1B-Instruct.
- We verify this approach across three datasets, two generator models, and two PRMs, and find that beam-search accuracy under the recalibrated PRM improves over the originally-calibrated PRM.
- We analyze accuracy gain and compute budget allocation by difficulty region, using self-consistency entropy as a measure of difficulty. We find that questions in the moderate-entropy region benefit most from the increased compute enabled by recalibration, while accuracy gains at the extreme entropy regions are negligible.

2 Related Work

2.1 PRM Calibration

PRMs (Lightman et al., 2024; Zhang et al., 2025; Ni et al., 2026) are trained on binarized correctness labels, which causes them to overestimate step-level success probabilities, particularly out-of-distribution (Cinquin et al., 2025). This miscalibration does not hurt *ranking*, but it does hurt beam search's ability to allocate test-time compute, which requires calibrated probabilities to decide when to expand versus terminate a path. Park et al. (2025b) and Ma et al. (2026) calibrate PRMs via quantile regression against Monte Carlo rollouts, but both use *unconditionally*-sampled trajectories for calibration — not steps conditioned on having survived search-induced pruning. This leaves the covariate shift induced by search itself uncorrected, which we show compounds with depth (§1); our method instead recalibrates using search-induced prefixes.

2.2 Search Algorithms for Math Reasoning

Test-time compute scaling (Snell et al., 2024) motivates methods ranging from sampling (self-consistency (Wang et al., 2022), Best-of-N (Wang et al., 2024)) to search-based approaches that prune incrementally using a verifier: beam search (Yao et al., 2023; Park et al., 2025a; Wang et al., 2025; Xie et al., 2023) and MCTS (Xu, 2023; Lin et al., 2025; Feng et al., 2023). Hu et al. (2025) anneal beam width with depth but do so uniformly rather than adaptively per-problem, while Park et al. (2025a) ensemble multiple policy models but still rely on uncalibrated PRM scores. Since self-verification without an external signal is unreliable (Huang et al., 2024), search quality depends on calibrated, instance-adaptive PRM guidance — the gap our work addresses.

2.3 Limitations of PRMs for Search

Prior work questions whether PRM-guided search delivers on this promise. Cinquin et al. (2025) find PRM-guided tree search does not reliably beat Best-of-N, with reliability degrading across datasets and policies. Dalal et al. (2026) and Joo & Klabjan (2026) show wider beams can reduce accuracy past an optimal width, attributing this to order-statistics bias in per-step argmax selection — modeled as constant across depth.

Our work differs along two axes: the shift we identify is *within*-dataset, *within*-policy, and *compounds with depth*, rather than cross-domain (Cinquin et al.) or depth-invariant (Dalal et al., Joo & Klabjan); and we verify it via a representational diagnostic (PRM hidden states, search-selected vs. unconditionally-sampled prefixes at matched depth) plus a depth-controlled causal comparison, neither present in this prior work. We view our contribution as complementary: where this literature documents *that* PRM-guided search underperforms, we identify survivorship-induced covariate shift as one concrete, testable mechanism.

3 PRM-Guided Beam Search Induces a Depth-Growing Survivorship Factor

During calibration, a PRM is trained on prefixes drawn from the unconditional generative distribution $P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x}) = \prod_{i=1}^{t} P(s_i \mid \boldsymbol{x}, s_{<i})$. At inference time under beam search, however, a prefix's presence in the beam reflects not only this generative likelihood but also whether every one of its constituent steps survived top-K pruning at its own depth. We show (Proposition 1, proof in Appendix A.1) that the resulting beam-search prefix distribution factorizes as

$$\pi_t \;=\; P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x}) \cdot \prod_{i=1}^{t} P\big(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}\big). \tag{1}$$

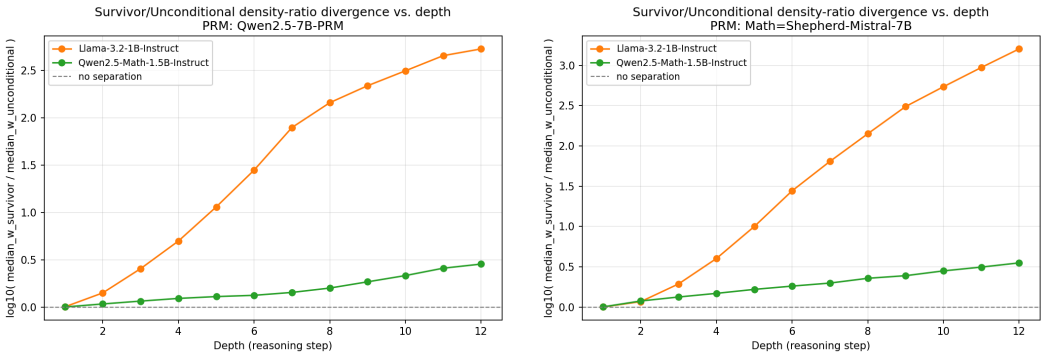
where the second factor — the *survivorship shrinkage* term — has no analog in the PRM's calibration data. Since each survival probability is at most 1, this factor is non-increasing in t and typically $\ll 1$ at greater depths, causing π_t to diverge increasingly from $P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x})$ as search proceeds. This is precisely the covariate shift the previous calibration procedure does not account for: the PRM is evaluated on prefixes drawn from π_t, but calibrated on prefixes drawn from $P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x})$, and the gap between the two compounds with depth.

We confirm this shift empirically in Figure 2: a classifier trained to distinguish the PRM's hidden-state representations of survivor-pool steps from unconditionally-sampled steps at matched depth achieves increasing discriminative power as depth grows, directly reflecting the growing divergence between π_t and $P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x})$ predicted by Proposition 1.

4 Methodology

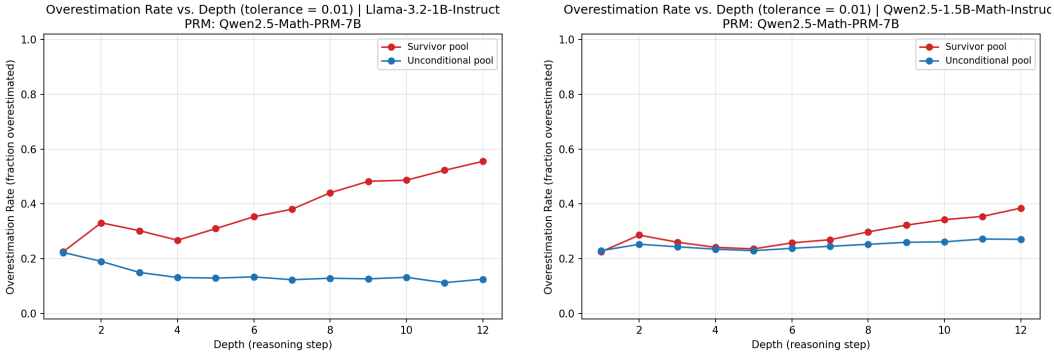
4.1 Survivor and Unconditional Prefix Collection

To capture the distribution a PRM actually encounters under beam search, we run live PRM-guided beam search over a set of training problems and record every surviving prefix at every depth. For each problem x, we generate reasoning steps one at a time using a fixed prompt template (raw



(a) Qwen2.5-Math-PRM (b) Math-Shepherd PRM

Figure 1: Survivor/unconditional density-ratio divergence vs. depth for (a) the Qwen PRM and (b) the Math-Shepherd PRM. The log-ratio of the median PRM-implied density on beam-survivor prefixes to the median density on unconditionally sampled prefixes grows monotonically with search depth for both PRMs and both evaluated policy models, empirically confirming the compounding out-of-distribution drift predicted by equation 1.

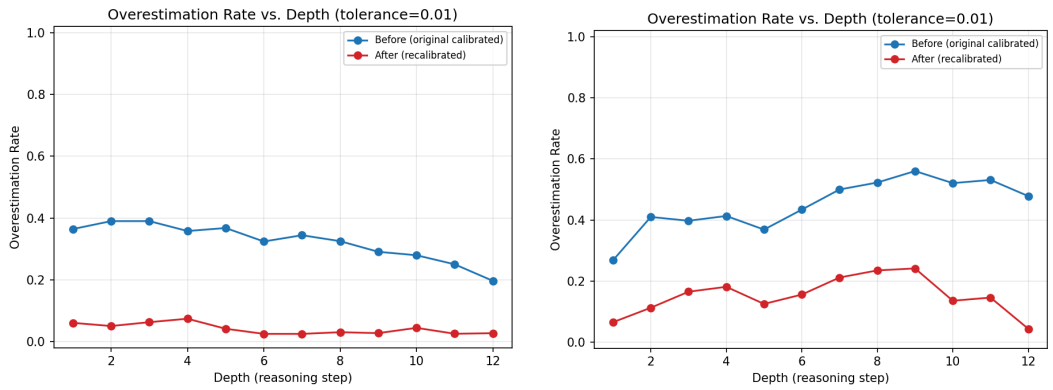


(a) Llama-3.2-1B-Instruct (b) Qwen2.5-Math-1.5B-Instruct

Figure 2: Overestimation rate (fraction of prefixes where $\hat{r}^{(\beta_{10})} - \tilde{p} > 0.01$) versus search depth, for the survivor pool and a matched unconditionally-sampled pool at equal depth, using Qwen2.5-Math-PRM-7B. For both generator policies, (a) Llama-3.2-1B-Instruct and (b) Qwen2.5-Math-1.5B-Instruct, the unconditional pool's overestimation rate remains roughly flat with depth, while the survivor pool's overestimation rate grows monotonically, confirming that the depth-dependent overestimation is driven by search-induced selection rather than prefix length or depth alone.

`llama-math-pf` / `qwen-math-pf` format, following Park et al. (2025b)), with `"\n\n"` and the end-of-turn token as stop conditions so each call produces exactly one step. At each depth t, every surviving beam is expanded by a fixed branch factor; candidates are scored by the PRM (§4.2), and only the top-K are retained, where K is the beam width. Every retained (*survivor*) prefix is recorded with its depth and PRM score $\hat{r}^{(\beta_{10})}$ (10th-quantile prediction). Search terminates once every surviving beam has a boxed answer or a maximum depth is reached. For every retained prefix, we estimate its true success probability $\tilde{p}$ via Monte Carlo rollout: K_{roll} independent continuations (policy model, fixed temperature, no PRM guidance), with $\tilde{p}$ the fraction reaching the correct answer. This process is run separately per (generator, PRM) pair and is checkpointed at the problem level.

To isolate survivorship selection from confounds like prefix depth or length, we construct a matched *unconditional* pool using an identical pipeline but with no PRM scoring or pruning: N independent complete paths are sampled per problem, each clipped at every step boundary to yield one prefix record per depth traversed. Each prefix is labeled with $\tilde{p}$ via the same rollout procedure, but is not PRM-scored at collection time. Since both pools share the same problems and generation pipeline,



(a) Math-Shepherd PRM

(b) Qwen2.5-Math-PRM

Figure 3: Overestimation rate (tolerance $= 0.01$) versus depth for the survivor pool, before and after on-policy recalibration, on held-out test-set prefixes, for Llama-3.2-1B-Instruct with (a) Math-Shepherd and (b) Qwen2.5-Math-PRM. In both cases, recalibration substantially reduces the overestimation rate at every depth, confirming the recalibrated PRM directly addresses the depth-dependent miscalibration identified in Figure 2.

differing only in whether PRM-based pruning occurred, comparing them at matched depth isolates the effect of survivorship selection specifically (§3).

4.2 PRM Scoring

We evaluate two PRMs with different training methodologies, Qwen2.5-Math-PRM-7B (Zhang et al., 2025) and Math-Shepherd-Mistral-7B (Wang et al., 2024), each via its publicly available quantile-regression-calibrated checkpoint from Park et al. (2025b). Both checkpoints use a 3-output quantile regression head (10th/50th/90th percentiles), fine-tuned via weighted pinball loss against Monte Carlo rollout labels. We use the 10th-percentile prediction $\hat{r}^{(\beta_{10})}$ throughout, as a conservative lower bound appropriate for budget-allocation decisions. To score an already-collected prefix, we pass the question and the `"\n\n"`-joined prefix text to the PRM's scoring function and take the score at the *final* step, following Park et al. (2025b).

4.3 On-Policy Recalibration

Given a survivor prefix pool for a (PRM, generator) pair, we further fine-tune the already-calibrated PRM checkpoint using the same quantile-regression objective as the original calibration (Park et al., 2025b), but training on *survivorship-conditioned* prefixes rather than unconditionally-sampled ones.

Bin-token discretization. Following Park et al. (2025b), each prefix's empirical success probability $\tilde{p}$ is discretized into one of B accuracy bins, each a special token placed after the PRM's step marker; the training loss is masked to this single position. The quantile head outputs three values $\hat{r}^{(\beta_n)}$, $\beta_n \in \{0.1, 0.5, 0.9\}$, compared against the bin's target via the weighted pinball loss:

$$\mathrm{wQL}(\hat{r}, \tilde{p}) \triangleq \frac{1}{N_q} \sum_{n=1}^{N_q} \left[\beta_n \cdot \max(0, \tilde{p} - \hat{r}^{(\beta_n)}) + (1 - \beta_n) \cdot \max(0, \hat{r}^{(\beta_n)} - \tilde{p}) \right].$$

We use B=9 bins, coarser than the original calibration's B=100, despite a higher-resolution rollout budget (K_{roll}=16 vs. 8); this is a deliberate simplification of the fine-tuning target space, consistent with recalibration as a lightweight targeted adjustment rather than full retraining. We leave the bin-count tradeoff to future work.

We fine-tune via LoRA on top of the calibrated checkpoint, with a substantially lower learning rate than the original calibration. The train/validation split is performed at the problem level to avoid

leaking correlated prefixes, and we select the final checkpoint by early stopping, since validation loss increases well before the configured training budget is exhausted.

4.4 Beam Search Evaluation

We evaluate downstream accuracy using PRM-guided beam search under two budget protocols, following the instance-adaptive scaling (IAS) framework of Park et al. (2025b). **Adaptive budget**: at each depth, given the surviving beam(s)' calibrated $q10$ score(s), the number of sampled continuations is $N_{\text{IAS}}(p, C) = \frac{\log(1-C)}{\log(1-p)}$, the minimum samples needed to reach confidence C that at least one succeeds, given per-sample success probability p; we use the most conservative surviving prefix's estimate, capped at $N_{\max}$. **Fixed budget**: a non-adaptive reference in which every surviving prefix is expanded by exactly $N_{\max}$ continuations regardless of confidence, isolating adaptive allocation from the PRM's ranking quality alone.

In both protocols, search terminates early once all surviving candidates have a boxed answer; final selection among remaining candidates uses the highest PRM score. We report accuracy plus mean generation calls and tokens per problem as an inference-cost measure, evaluating both protocols with both the baseline and recalibrated checkpoints, across two generator policies (Llama-3.2-1B-Instruct, Qwen2.5-Math-1.5B-Instruct) and both PRMs, on MATH500.

5 Experiments

Process reward models. We evaluate two PRMs with different training methodologies: **Qwen2.5-Math-PRM-7B** (Zhang et al., 2025), trained on consensus-filtered labels combining human and automated annotation, and **Math-Shepherd-Mistral-7B** (Wang et al., 2024), trained on fully automated Monte-Carlo-rollout labels. For both, we use the publicly released quantile-regression-calibrated checkpoints of Park et al. (2025b) as our starting point (§4.2), and refer to these as the *baseline* (originally calibrated) PRMs throughout.

Generator policies. We generate reasoning steps and estimate empirical success probabilities using two policy models of different scale and specialization: **Llama-3.2-1B-Instruct** (Touvron et al., 2023) and **Qwen2.5-Math-1.5B-Instruct** (Yang et al., 2024). Each PRM is calibrated separately per policy model (§4.2), and we recalibrate (§4.3) each of the four resulting (PRM, policy) pairs independently.

Datasets. We use **MATH500** (Hendrycks et al., 2021) as our primary evaluation benchmark, spanning five annotated difficulty levels, and use its training split (levels 1–5) to collect survivor and unconditional prefixes for our diagnostic analysis (§4.1) and recalibration (§4.3). We additionally use **tinyGSM8K** (Maia Polo et al., 2024), a reduced-size variant of GSM8K (Cobbe et al., 2021). As a further test of generalization to harder problems, we report an additional analysis on **AIME24** and **AIME25**, a substantially more difficult and much smaller benchmark (30 problems per year), noting the correspondingly larger variance in these results due to sample size.

Survivor and unconditional prefix collection. For each (PRM, policy) pair, we collect survivor prefixes via live beam search with beam width $K{=}4$ and branch factor 4, to a maximum depth of 30 steps, over 400 problems from each of MATH500's five training-split difficulty levels (2000 problems total). Each surviving prefix's empirical success probability $\tilde{p}$ is estimated via $K_{\text{roll}}{=}16$ Monte Carlo rollouts. We collect a matched unconditional prefix pool using an identical generation pipeline with no PRM scoring or pruning (§4.1), over the same problem set. Full generation hyperparameters (temperature, top-p, max tokens) are listed in Appendix A.3.

On-policy recalibration. We fine-tune each baseline (PRM, policy) checkpoint on its corresponding survivor prefix pool following the procedure in §4.3, using a LoRA adapter continued from the already-calibrated checkpoint's weights and a substantially lower learning rate than the original calibration. We select the final checkpoint via early stopping on a held-out validation split of problems (not rows), since validation loss begins increasing well before the configured training budget is exhausted. Full training hyperparameters are listed in Appendix A.3. We refer to the resulting checkpoints as *finetuned* (recalibrated) throughout.

Table 1: MATH500: accuracy (%) across methods, generators, and PRMs.

	Qwen		Llama	
Method	**Qwen-PRM**	**Math-Shepherd**	**Qwen-PRM**	**Math-Shepherd**
SC@20	79.8	**79.8**	31.4	31.4
BoN@20	**84.2**	77.4	32.4	27.8
Unconditional (K=2, $N_{\max}$=8)	77.4	69.8	47.6	32.4
Recalibrated (K=2, $N_{\max}$=8)	79.0	70.2	49.0	34.4
Unconditional (K=8, $N_{\max}$=16)	78.4	73.6	60.8	**40.0**
Recalibrated (K=8, $N_{\max}$=16)	81.4	72.0	**65.2**	39.8

Table 2: Generalization check on AIME24+25 (n=60), broad width, C=0.90.

PRM	**Generator**	**Baseline Acc (%)**	**Finetuned Acc (%)**	**Δ**	**Gen. calls ratio**
Qwen-PRM	Qwen	13.3	**15.0**	+1.7	1.826×
Qwen-PRM	Llama	3.3	**6.7**	+3.3	0.955×
Math-Shepherd	Qwen	8.3	**10.0**	+1.7	1.761×
Math-Shepherd	Llama	0.0	**0.0**	0.0	1.150×

Beam search evaluation. We evaluate baseline and finetuned checkpoints on MATH500's 500-problem test split using PRM-guided beam search under two protocols (§4.4): a *fixed* budget, in which every surviving prefix is expanded by exactly $N_{\max}$ continuations at every depth, and an *adaptive* budget following the instance-adaptive scaling (IAS) framework of Park et al. (2025b), with target confidence C=0.90 and quantile β=0.1. We evaluate both protocols at two beam-width configurations: a *narrow* setting (K=2, $N_{\max}$=8) and a *broad* setting (K=8, $N_{\max}$=16). Note that this evaluation-time beam width does not match the fixed beam width used during survivor prefix collection (K=4, branch factor 4).

Self-consistency and Best-of-N baselines. To contextualize beam search against alternative inference-time scaling methods, we additionally evaluate plain self-consistency (majority vote over N=20 independently sampled completions) and PRM-guided Best-of-N (max-of-min step-score aggregation over the same 20 samples) on both MATH500 and tinyGSM8K, for both policy models, using the baseline PRM checkpoints. Full details are given in §6.5.

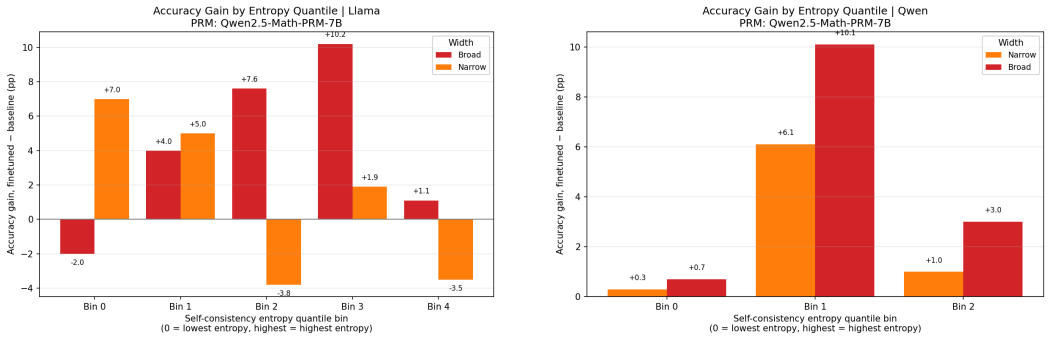
6 Results and Analysis

6.1 Downstream Task Performance

Table 1 reports MATH500 accuracy across all methods, generators, and PRMs. TinyGSM results are available at Appendix (Table 5). Beam search with the originally-calibrated PRM already outperforms self-consistency and PRM-guided Best-of-N for Llama at both widths and both PRMs. Recalibration improves accuracy in six of eight (PRM, generator, width) configurations: at broad width, Qwen-PRM improves both generators (Qwen: $0.7840 \rightarrow 0.8140$; Llama: $0.6080 \rightarrow 0.6520$, the largest single gain at $+4.40$ percentage points), while Math-Shepherd is mixed (Llama narrow width: $0.3240 \rightarrow 0.3440$; Qwen broad width: $0.7360 \rightarrow 0.7200$). For the Qwen generator, sampling techniques such as self-consistency and Best-of-N perform better because of its questions skewed towards highly certain regions or low entropy regions, which we will discuss in section 6.5.

6.2 Generalization to Easier and Harder Benchmarks

We test generalization on tinyGSM8K (easier) and AIME24+25 (harder, evaluated only at broad width due to compute constraints).



(a) Llama-3.2-1B (b) Qwen2.5-Math-1.5B

Figure 4: Accuracy gain (finetuned − baseline) by self-consistency entropy quantile bin, comparing broad (K=8, $N_{\max}$=16) and narrow (K=2, $N_{\max}$=8) search width, for Qwen-PRM. Bin 0 is the lowest-entropy (highest self-consistency agreement) quantile. For Llama (a), the entropy region of peak benefit shifts and partially reverses sign between widths; for Qwen (b), the same qualitative shape (peak at the moderate-entropy bin) holds at both widths, with larger magnitude at broad width.

Table 3: Ratio of fixed-budget to adaptive-budget compute (generator calls) by entropy bin, Qwen-PRM, broad width. Bin 0 is lowest entropy, Bin 4 highest.

		Entropy bin				
Generator	**Checkpoint**	**0 (low)**	**1**	**2**	**3**	**4 (high)**
Llama	Baseline	2.27×	1.55×	1.27×	1.28×	1.12×
Llama	Finetuned	1.92×	1.21×	1.08×	1.12×	1.09×
Qwen	Baseline	6.42×	3.85×	3.92×	—	—
Qwen	Finetuned	4.34×	2.22×	1.90×	—	—

On tinyGSM8K (Table 5), recalibration improves Qwen under both PRMs at broad width ($0.87 \rightarrow 0.92$; Math-Shepherd: $0.84 \rightarrow 0.93$) and Llama under Qwen-PRM at both widths (narrow: $0.63 \rightarrow 0.70$; broad: unchanged). Math-Shepherd with Llama shows a small broad-width decline ($0.78 \rightarrow 0.77$), the only regression on this benchmark.

On AIME24+25 (n=60, Table 2), recalibration improves both generators under Qwen-PRM ($13.33\% \rightarrow 15.00\%$; $3.33\% \rightarrow 6.67\%$) and Qwen under Math-Shepherd ($8.33\% \rightarrow 10.00\%$). Llama with Math-Shepherd solves 0/60 problems under either checkpoint, reflecting a capability floor for this generator on this benchmark rather than a null effect of recalibration. Overall, the MATH500 pattern extends directionally to both an easier and a substantially harder benchmark, excepting the lowest-capability (Llama, Math-Shepherd) combination.

6.3 Accuracy Gain by Entropy Region

We stratify MATH500 questions by self-consistency entropy (the Shannon entropy of the answer distribution across N=20 independently sampled completions) into quantile bins — 5 for Llama, 3 for Qwen, since roughly 60% of Qwen's questions share near-zero entropy — and compute the recalibration accuracy gain within each bin (Figure 4). Across both generators and both search widths, the gain is consistently smallest, and sometimes negative, at the lowest-entropy bin, and largest at a moderate-entropy bin (up to +10.2pp for Llama, +10.1pp for Qwen), indicating recalibration benefits questions of partial, resolvable difficulty most.

6.4 Compute Cost by Entropy Region

We next examine how search-time compute is allocated across these same entropy regions. Table 3 reports the ratio of fixed-budget to adaptive-budget compute (generation calls) within each entropy

Table 4: Accuracy (%) by entropy bin for all methods on MATH500, Qwen-PRM, broad width. Bold marks the best non-oracle method in each row.

		Accuracy (%)				
Generator	**Entropy bin**	**SC**	**PRM-BoN**	**Beam (base)**	**Beam (FT)**	**Pass@20**
Llama	0 (low)	68.0	63.0	**87.0**	85.0	86.0
Llama	1	36.0	42.0	67.0	**71.0**	78.0
Llama	2	32.4	28.6	60.0	**67.6**	63.8
Llama	3	13.0	17.6	51.9	**62.0**	45.4
Llama	4 (high)	5.7	9.2	35.6	**36.8**	20.7
Qwen	0 (low)	**97.3**	**97.3**	95.3	96.0	97.3
Qwen	1 (mid)	73.7	**83.8**	70.7	80.8	92.9
Qwen	2 (high)	33.0	**45.0**	35.0	38.0	71.0

bin, for both checkpoints and both generators, at broad width. This ratio decreases monotonically with entropy across every configuration: for Llama, fixed budget spends $2.27\times$ more compute than adaptive at the lowest-entropy bin, narrowing to $1.12\times$ ($1.09\times$ for the finetuned checkpoint) at the highest-entropy bin; for Qwen, the corresponding ratios shrink from $6.42\times$ ($4.34\times$ finetuned) at the lowest entropy to $1.90\times$ at the highest.

This confirms that instance-adaptive budgeting concentrates its compute savings specifically on low-entropy (easy) questions, where extensive search is unnecessary, while allocating compute much closer to a fixed, brute-force budget on moderate-to-high-entropy questions, where genuine difficulty justifies the additional expenditure. This pattern holds for both the originally-calibrated and the recalibrated PRM, indicating that recalibration sharpens *how much* budget is allocated where, rather than altering this underlying difficulty-proportional allocation strategy itself.

6.5 Accuracy by Entropy Region Across Methods

Finally, we compare all five methods — self-consistency (SC), PRM-guided Best-of-N (PRM-BoN), beam search with the baseline PRM, beam search with the recalibrated PRM, and the Pass@20 — within each entropy region, for Qwen-PRM at broad width (Table 4). For Llama, beam search (both checkpoints) outperforms SC and PRM-BoN at every entropy level, with the gap widening at higher entropy (Table 4): at the highest-difficulty bin, SC and PRM-BoN reach only 5.7% and 9.2%, versus 35.6% (baseline) and 36.8% (recalibrated) for beam search. For Qwen, PRM-BoN matches or exceeds beam-baseline at moderate-to-high entropy (83.8% vs. 70.7% at moderate entropy), and only the recalibrated beam search (80.8%) closes most of this gap. This suggests search guidance is most valuable when self-consistency signal is weak (Llama); when it is already strong (Qwen), Best-of-N captures much of the same benefit, and recalibration mainly helps beam search catch up.

7 Conclusion

We showed that PRM-guided beam search induces a self-referential covariate shift: because a prefix survives to be scored only if every preceding step survived pruning, PRMs are evaluated on a distribution they themselves shaped, one that diverges increasingly from the unconditionally-sampled prefixes used for calibration. This causes calibrated PRMs to grow overconfident with search depth, even after standard calibration. Our on-policy recalibration, which fine-tunes on the PRM's own search-induced prefixes, substantially reduces this overestimation and improves downstream beam-search accuracy with Qwen-PRM, with gains concentrated among moderate-uncertainty questions; results with Math-Shepherd are more mixed. These findings suggest PRM calibration should be treated as coupled to the search procedure it supports, rather than as a fixed, transferable property, and we hope this motivates further work on selection-aware calibration for verifier-guided search.

AI USE STATEMENT

In this work, we used generative AI tools for writing assistance, including improving clarity, grammar, coherence, and conciseness; reducing the word count to meet page limits; and reformatting tables. We also used generative AI tools to assist with the discovery of relevant related work and to produce initial drafts of portions of the Introduction, Related Work, and experimental setup. These drafts were generated from detailed outlines, notes, and results provided by the authors.

All AI-assisted content was reviewed by the authors. For related work, every identified paper was independently read and verified by the authors, and citations were taken directly from the authors' BibTeX file rather than generated by AI tools. For AI-assisted writing, the authors checked each passage for technical accuracy and consistency with the methods and experimental results, substantially revised the drafts, and rewrote them where necessary. Generative AI tools were not used to develop the research ideas, methodology, experimental design, implementation, analysis, or interpretation of results. These aspects of the work are entirely the authors' own.

We take full responsibility for the final content of this work, including all text, claims, analyses, and results produced with the assistance of generative AI tools.

REPRODUCIBILITY STATEMENT

For implementation, we have shared an anonymous code base that can be used to reproduce the results. We have also discussed the hyperparameters that we used in the appendix section.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396, 2023.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *Advances in Neural Information Processing Systems*, 37:27689–27724, 2024.

Tristan Cinquin, Geoff Pleiss, and Agustinus Kristiadi. Limits of prm-guided tree search for mathematical reasoning with llms. *arXiv preprint arXiv:2510.20272*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL `https://arxiv.org/abs/2110.14168`.

Gal Dalal, Assaf Hallak, Gal Chechik, and Yftah Ziser. More test-time compute can hurt: Overestimation bias in llm beam search. *arXiv preprint arXiv:2603.15377*, 2026.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 8154–8173, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Pengfei Hu, Zhenrong Zhang, Qikai Chang, Shuhang Liu, Jiefeng Ma, Jun Du, Jianshu Zhang, Quan Liu, Jianqing Gao, Feng Ma, et al. Prm-bas: Enhancing multimodal reasoning through prm-guided beam annealing search. *arXiv preprint arXiv:2504.10222*, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

Taejong Joo and Diego Klabjan. Mitigating over-optimization in prm-guided search in mathematical reasoning by optimizing the guide. *arXiv preprint arXiv:2608.30051*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison 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*, volume 2024, pp. 39578–39601, 2024.

Qingwen Lin, Boyan Xu, Zijian Li, Zhifeng Hao, Keli Zhang, and Ruichu Cai. Cmcts: A constrained monte carlo tree search framework for mathematical reasoning in large language model. *Applied Intelligence*, 56, 2025. doi: 10.1007/s10489-025-07044-6.

Rachel Ma, Dylan Hadfield-Menell, and Kristjan Greenewald. Distributional process reward models: Calibrated prediction of future rewards via conditional optimal transport. *arXiv preprint arXiv:2605.06785*, 2026.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinyBenchmarks: evaluating LLMs with fewer examples. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 34303–34326. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/maia-polo24a.html.

Jingwei Ni, Ekaterina Fadeeva, Tianyi Wu, Mubashara Akhtar, Jiaheng Zhang, Elliott Ash, Markus Leippold, Timothy Baldwin, See Kiong Ng, Artem Shelmanov, et al. Efficient test-time scaling of multi-step reasoning by probing internal states of large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11667–11689, 2026.

Sungjin Park, Xiao Liu, Yeyun Gong, and Edward Choi. Ensembling large language models with process reward-guided tree search for better complex reasoning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10256–10277, 2025a.

Young-Jin Park, Kristjan Greenewald, Kaveh Alim, Hao Wang, and Navid Azizan. Know what you don't know: Uncertainty calibration of process reward models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=hzMkfIrdDT.

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.

Amir Taubenfeld, Tom Sheffer, Eran Ofek, Amir Feder, Ariel Goldstein, Zorik Gekhman, and Gal Yona. Confidence improves self-consistency in LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20090–20111, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1030. URL https://aclanthology.org/2025.findings-acl.1030/.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

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. *arXiv preprint arXiv:2203.11171*, 2022.

Yi Wang, Junxiao Liu, Shimao Zhang, Jiajun Chen, and Shujian Huang. Pats: Process-level adaptive thinking mode switching. *arXiv preprint arXiv:2505.19250*, 2025.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36:41618–41650, 2023.

Haotian Xu. No train still gain. unleash mathematical reasoning of large language models with monte carlo tree search guided by energy function. *ArXiv*, abs/2309.03224, 2023. doi: 10.48550/arxiv.2309.03224.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement, 2024. URL `https://arxiv.org/abs/2409.12122`.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

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. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025.

A Appendix

A.1 Out of distribution with depth

Proposition 1 (Survivorship decomposition of the beam-search prefix distribution)**.** *For all $t \geq 1$,*

$$\pi_t \;=\; P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x}) \cdot \prod_{i=1}^{t} P\big(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}\big),$$

where π_t denotes the probability that prefix $s_{1:t}$ is present in the beam at depth t under beam search, $P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x}) = \prod_{i=1}^{t} P(s_i \mid \boldsymbol{x}, s_{<i})$ is the unconditional autoregressive (chain-rule) likelihood of the prefix under the policy model, and $P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i})$ is the probability that step s_i survives top-K pruning at depth i (Equation 1).

Proof. We proceed by induction on t.

Base case ($t = 1$). By definition, $\pi_1 = \pi_0 \cdot P(s_1 \mid \boldsymbol{x}) \cdot P(\mathrm{surv}(s_1) \mid \boldsymbol{x}, s_1)$, where $\pi_0 = 1$, since the empty prefix trivially survives with probability 1 before any pruning has occurred. Thus

$$\pi_1 = P(s_1 \mid \boldsymbol{x}) \cdot P(\mathrm{surv}(s_1) \mid \boldsymbol{x}, s_1) = P_{\mathrm{LM}}(s_{1:1} \mid \boldsymbol{x}) \cdot \prod_{i=1}^{1} P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}),$$

since $P_{\mathrm{LM}}(s_{1:1} \mid \boldsymbol{x}) = P(s_1 \mid \boldsymbol{x})$ by the chain rule. The claim holds for $t = 1$.

Inductive step. Assume the claim holds at depth $t - 1$:

$$\pi_{t-1} = P_{\mathrm{LM}}(s_{1:t-1} \mid \boldsymbol{x}) \cdot \prod_{i=1}^{t-1} P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}). \tag{IH}$$

By the recursive definition of π_t,

$$\pi_t = \pi_{t-1} \cdot P(s_t \mid \boldsymbol{x}, s_{<t}) \cdot P(\mathrm{surv}(s_t) \mid \boldsymbol{x}, s_{1:t}).$$

Substituting (IH),

$$\pi_t = \left[P_{\mathrm{LM}}(s_{1:t-1} \mid \boldsymbol{x}) \cdot \prod_{i=1}^{t-1} P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}) \right] \cdot P(s_t \mid \boldsymbol{x}, s_{<t}) \cdot P(\mathrm{surv}(s_t) \mid \boldsymbol{x}, s_{1:t}).$$

Regrouping the generative terms via the chain rule, $P_{\mathrm{LM}}(s_{1:t-1} \mid \boldsymbol{x}) \cdot P(s_t \mid \boldsymbol{x}, s_{<t}) = P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x})$, and regrouping the survival terms,

$$\prod_{i=1}^{t-1} P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}) \cdot P(\mathrm{surv}(s_t) \mid \boldsymbol{x}, s_{1:t}) = \prod_{i=1}^{t} P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}),$$

we obtain

$$\pi_t = P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x}) \cdot \prod_{i=1}^{t} P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}),$$

which is the claim at depth t. □

Since each survival factor $P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i}) \in (0, 1]$ under top-K pruning, with equality only if every candidate proposed at depth i survives (i.e., no pruning occurs at that depth), the product $\prod_{i=1}^{t} P(\mathrm{surv}(s_i) \mid \boldsymbol{x}, s_{1:i})$ is non-increasing in t: each additional depth introduces one further factor no greater than 1. Consequently, whenever pruning is active at every depth, π_t diverges increasingly from $P_{\mathrm{LM}}(s_{1:t} \mid \boldsymbol{x})$ as t grows, without requiring any further assumption on the specific form of the PRM's selection rule.

A.2 Downstream performance

Table 5: TinyGSM8K: accuracy across methods, generators, and PRMs.

	Qwen		**Llama**	
	Qwen-PRM	**Math-Shepherd**	**Qwen-PRM**	**Math-Shepherd**
SC@20	0.92	0.92	0.32	0.32
BoN@20	**0.95**	0.91	0.36	0.32
Unconditional (K=2, $N_{\max}$=8)	0.86	0.84	0.63	0.69
Recalibrated (K=2, $N_{\max}$=8)	0.87	0.88	0.70	0.65
Unconditional (K=8, $N_{\max}$=16)	0.87	0.84	0.83	**0.78**
Recalibrated (K=8, $N_{\max}$=16)	0.92	**0.93**	**0.83**	0.77

A.3 Hyperparameters

Table 6: Prefix collection hyperparameters.

Parameter	**Value**
Beam width (K)	4
Branch factor	4
Maximum beam depth	30
Monte Carlo rollouts per prefix (K_{roll})	16
Problems per difficulty level	400
Difficulty levels used	1–5 (all)
Total problems (collection)	2000
Data split used for collection	MATH500 training split
Maximum tokens per generated step	256