

A Process Supervision Approach to Correcting Logical Errors in On-Device Language Models

Devang Borkar (dnborkar@ucdavis.edu)
Kwabena Manu (dkmanu@ucdavis.edu)
Linus Lam (ytllam@ucdavis.edu)

November 24 2025

Abstract

Large Language Models (LLMs) frequently struggle with multi-step mathematical reasoning due to logical hallucinations, where a single intermediate error propagates to the final answer. To mitigate this problem, recent works have demonstrated that Process Supervision, where each step in the reasoning chain is verified, provides better guidance and higher accuracy than Outcome Supervision, where only the final answer in the reasoning chain is verified, and other simple consensus methods like Self-Consistency. Using Process Supervision, where a verifier model, is trained using a step-level annotated dataset, imposes a prohibitively high annotation cost. It still remains an open question, however, whether this upfront high investment yields a positive return for small, compute-constrained models (sub-10B parameter models), or whether using cheaper consensus-based methods is sufficient.

In this work, we benchmark the inference sample efficiency of PRMs in constrained-compute environments. We employ a lightweight generator model (Qwen3-0.6B) to generate candidate reasoning chains for mathematical problems from the GSM8K dataset, and benchmark a lightweight PRM verifier created from the Qwen3-8B model and finetuned on the PRM800K dataset. We evaluate the PRM against Self-Consistency baseline and also compare search strategies, finding that Best-of-N (58.2% accuracy) outperforms step-wise Beam Search (53.8% accuracy) for these small models. We demonstrate that with a PRM with Best-of-N strategy, we can significantly outperform the self-consistency baseline, achieving a 7% increase in accuracy from 33.3% to 40.4%. Again, we show that best-of-N sampling achieves higher accuracy than a beam search with the same rough compute cost, and that PRM yields higher accuracy than self-consistency across a range of compute budgets. These results demonstrate that for constrained environments, investing in step-level annotation is justified as it increases overall accuracy and drastically reduces the inference compute needed to reach correct solutions.

1 Introduction

Large Language Models (LLMs) have demonstrated impressive capabilities in mathematical reasoning, largely enabled by methods like Chain-of-Thought (CoT) prompting [7], which allow the LLMs to express its reasoning process in getting to a final answer. However, with these methods, LLMs often produce hallucinations, wherein they are able to reach the right answers even with flawed logic in their reasoning steps or have a wrong logical step derail the entire solution. This is particularly problematic in using LLMs for mathematical reasoning, as we are usually not only interested in the final answers in mathematics, but also the logic in intermediate steps as well. To address this, researchers in this field have experimented with and adopted Process Supervision, where a Process Reward Model (PRM) is trained to verify the reasoning step-by-step in LLMs. This approach has been shown to significantly outperform Outcome Supervision, which is only interested in verifying the final answer of a reasoning chain [4, 5].

While the superior performance of Process Reward Models (PRMs) is well-established for large models, a critical economic question remains unanswered for small-scale, on-device models (sub-10B parameters). Training a PRM requires expensive, time-consuming step-level human annotations, such as the PRM800K dataset [4], while outcome-based methods like Self-Consistency (Majority

Voting) [6] require no specially annotated data. For researchers and developers working in resource-constrained environments, it is unclear if this high annotation cost yields a sufficient return on investment (ROI). We therefore ask, does a small, weak verifier (PRM) actually have the capacity to guide a small generator, or is the investment in granular labeling worth the trouble?

In this work, we quantify the return on investment (ROI) of process supervision for small models. We simulate the high-investment strategy by training a small Process Reward Model as a verifier by finetuning a Qwen3-8B model [8] on a step-level annotated mathematics dataset, PRM800K [4]. We then pair it with a lightweight generator model (Qwen3-0.6B) [8] that is used to generate candidate chain-of-thought solutions to various mathematical problems in our experiments. We evaluate the inference sample efficiency of our lightweight PRM against two established baselines: Greedy Decoding and Self-Consistency. Furthermore, to determine the optimal inference strategy for resource-constrained devices, we extend our analysis to compare the standard Best-of-N approach against a memory-efficient step-wise Beam Search. Our experiments on the GSM8K dataset [1] demonstrate that the upfront cost of doing step-level annotation in the lightweight PRM yields direct improvements in accuracy compared to the baselines with reduced inference compute.

2 Methodology

2.1 Problem formalization

We frame mathematical reasoning as a sequential decision making process. Given a problem x , the goal is to generate a solution chain y that is a sequence of discrete steps such that

$$y = (s_1, s_2, \dots, s_T) \quad (1)$$

The correctness of the final answer depends on the validity of each preceding step. Our system divides this task into 2 separate components:

1. Generator Policy ($\pi_\theta(y|x)$): A small language model used for exploring the solution space and generating candidate reasoning chains.
2. Process Reward Model i.e Verifier ($V_\theta(s_t|x, s_{<t})$): A more capable language model to score the correctness of step s_t based on the problem x and the history of previous steps $s_{<t}$.

2.2 Training the Process Reward Model

The verifier is trained on the PRM 800K dataset [4] consisting of mathematical reasoning problems such that after filtering out neutral annotations, each step label is binarized as $l_t^{(i,j)} \in \{0, 1\}$, where $l_t^{(i,j)} = 1$ denotes a correct step and $l_t^{(i,j)} = 0$ denotes an incorrect step.

We start from the pre-trained Qwen3-8B [8] language model and augment it with a simple binary classification head. Let $h_t^{(i,j)} \in \mathbb{R}^d$ denote the hidden state of the final token of step $s_t^{(i,j)}$ for a given trajectory. The classification head is a linear layer that maps this hidden state to a scalar logit

$$z_t^{(i,j)} = V_\theta(s_t^{(i,j)} | c_t^{(i,j)}) \in \mathbb{R}, \quad (2)$$

where V_θ denotes the verifier network with parameters θ , and $c_t^{(i,j)}$ is the context for the current step. The context $c_t^{(i,j)}$ consists of the original problem statement $x^{(j)}$ and the history of preceding reasoning steps $s_{<t}^{(i,j)}$ in the same trajectory:

$$c_t^{(i,j)} = (x^{(j)}, s_{<t}^{(i,j)}). \quad (3)$$

The corresponding probability of correctness for step $s_t^{(i,j)}$ can be obtained by applying a sigmoid to the logit $z_t^{(i,j)}$,

$$p_t^{(i,j)} = \sigma(z_t^{(i,j)}) = \frac{1}{1 + \exp(-z_t^{(i,j)})}, \quad (4)$$

To fine-tune this 8-billion-parameter model efficiently, we use Low-Rank Adaptation (LoRA) [3]. Instead of updating all pre-trained weights, LoRA freezes the original weight matrix W_0 of a given linear transformation and injects a pair of trainable low-rank matrices A and B of rank r . Let $x \in \mathbb{R}^{d_{\text{in}}}$ be the input activation to such a layer and $h \in \mathbb{R}^{d_{\text{out}}}$ its output. With LoRA, the forward pass is modified as

$$h = W_0x + \frac{\alpha}{r}BAx, \quad (5)$$

where $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is the frozen pre-trained weight matrix, $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are the trainable low-rank matrices, r is the LoRA rank, and α is a scaling factor. For our experiments, we apply LoRA to the query, key, value, and output projection matrices in the attention layers, using a rank of $r = 16$, a scaling factor of $\alpha = 32$, and a LoRA dropout rate of 0.1. This setup significantly reduces the number of trainable parameters while still allowing the model to adapt effectively to the verification task.

Optimization objective: Let \mathcal{P} denote the set of training problems, and let $|\mathcal{P}|$ be the number of distinct problems. For each problem $j \in \{1, \dots, |\mathcal{P}|\}$, we sample M_j solution trajectories. For trajectory $i \in \{1, \dots, M_j\}$ of problem j , we obtain a sequence of $T_{i,j}$ reasoning steps

$$\{s_t^{(i,j)}\}_{t=1}^{T_{i,j}}, \quad (6)$$

with corresponding binary labels $\{l_t^{(i,j)}\}_{t=1}^{T_{i,j}}$, where $l_t^{(i,j)} \in \{0, 1\}$ indicates whether step $s_t^{(i,j)}$ is correct.

The total number of labeled steps in the training corpus is

$$N = \sum_{j=1}^{|\mathcal{P}|} \sum_{i=1}^{M_j} T_{i,j}. \quad (7)$$

We train the verifier by minimizing the binary cross-entropy loss computed directly on logits. For a single labeled step $(s_t^{(i,j)}, l_t^{(i,j)})$ with logit $z_t^{(i,j)}$, the per-step loss is

$$\ell(z_t^{(i,j)}, l_t^{(i,j)}) = \log(1 + \exp(z_t^{(i,j)})) - l_t^{(i,j)} z_t^{(i,j)}. \quad (8)$$

The overall training objective is the average of this loss over all labeled steps:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{j=1}^{|\mathcal{P}|} \sum_{i=1}^{M_j} \sum_{t=1}^{T_{i,j}} [\log(1 + \exp(z_t^{(i,j)})) - l_t^{(i,j)} z_t^{(i,j)}], \quad (9)$$

where $z_t^{(i,j)} = V_\theta(s_t^{(i,j)} | c_t^{(i,j)})$ is the logit produced by the verifier for step $s_t^{(i,j)}$ conditioned on its context $c_t^{(i,j)}$, and $l_t^{(i,j)}$ is the corresponding binary target label.

Training setup. We optimize $\mathcal{L}(\theta)$ using the AdamW optimizer over 2 epochs with a cosine learning rate schedule. The initial learning rate is set to 2×10^{-4} after a warm-up phase covering 3% of the total training steps. To handle memory constraints while preserving a stable training signal, we use a per-device batch size of 4 and accumulate gradients for 4 steps, corresponding to an effective global batch size of 16. We apply a weight decay of 0.01 to all trainable parameters for regularization.

2.3 Inference Strategy: Process-Guided Best-Of-N

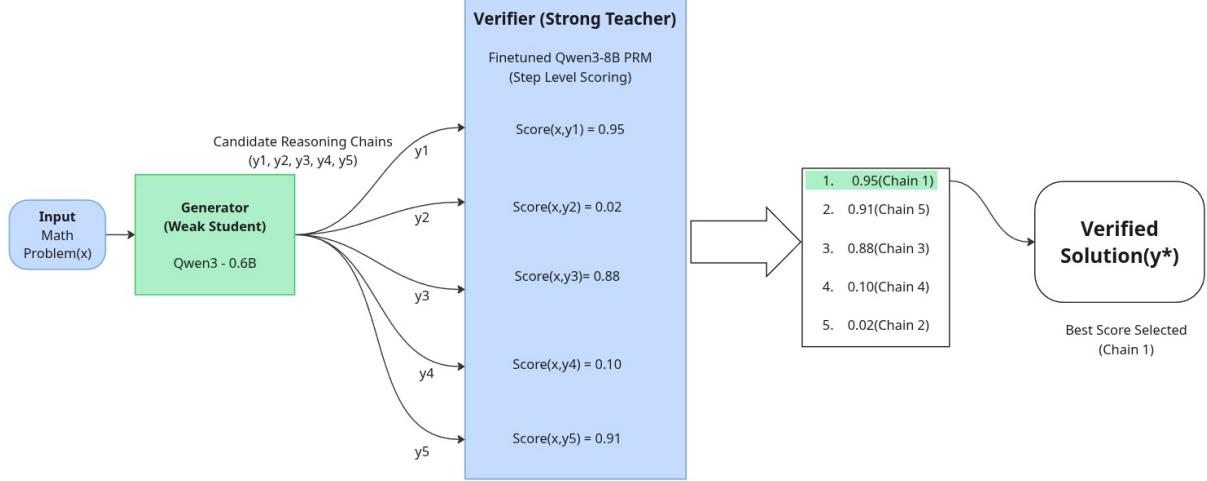


Figure 1: PRM-guided Inference Pipeline. Verified Solution (\hat{y})

During inference, we use the trained Process Reward Model (PRM) to evaluate and select the most plausible reasoning chain from a set of candidates generated by the smaller low-latency model. This process involves three distinct stages:

- 1. Generation:** For a given problem input z , the weak generator model (π_ϕ) produces a diverse set of N candidate reasoning chains, denoted as $\{y^{(1)}, y^{(2)}, \dots, y^{(N)}\}$. We use Qwen3-0.6B as our generator. Its small size allows for efficient sampling on resource constrained devices.
- 2. Verification:** The process verifier (V_θ) evaluates each individual step of every candidate chain. The final score for a complete chain $y^{(k)}$ is calculated as the joint probability of all its constituent steps being correct.

$$Score(y^{(k)}) = \prod_{t=1}^{T_k} V_\theta(s_t^{(k)} | x, s_{<t}^{(k)}) \quad (10)$$

where T_k is the total number of steps in chain $y^{(k)}$ and $s_t^{(k)}$ is the t -th step of that chain. Qwen3-8B is fine-tuned to act as this reward model.

- 3. Selection:** The final predicted solution \hat{y} is the reasoning chain that achieves the highest cumulative verification score among all generated candidates:

$$\hat{y} = \operatorname{argmax}_{y \in \{y^{(1)}, \dots, y^{(N)}\}} Score(y) \quad (11)$$

This strategy also allows filtering out reasoning paths that may lead to a correct final answer through flawed logic, prioritizing chains that are internally consistent and verified at a granular level.

3 Experiment setup

Our experiments will be performed on the test split of the GSM8K dataset, which consists of 1319 grade school-level math word problems and their solutions in natural language. These problems take 2-8 steps to solve, and involve concepts up to early algebra at most [1].

3.1 Experiment 1: Baseline comparisons

Solution evaluation method(s): Greedy, Self-consistency, PRM

Variable(s): Evaluation method

Measured: Solution accuracy

This experiment compares PRM-guided inference against self-consistency inference. Multiple solution chains are generated in a greedy fashion, i.e. the entire chain of thought is generated at once, and the majority vote of each chain’s final answer is taken as the output. We perform this experiment across all 1319 problems of the GSM8K test set. For a more robust comparison, we generate 16 candidate solutions for each problem, and reuse these candidates between the PRM and self-consistency inferences. As another point of reference, we also consider the accuracy rate of a single greedy inference. However, this is not directly comparable with the PRM or self consistency methods, as there is no consideration of multiple candidate solutions.

3.2 Experiment 2: Search method comparisons

Solution evaluation method(s): PRM

Variable(s): Search strategy (Best of N vs. Beam search)

Measured: Solution accuracy

This experiment investigates whether step-wise beam search performs better than the best-of-N chain-wise approach we use in Experiment 1. We choose beam search as it is a memory efficient, heuristic search algorithm, which is less compute intensive than other algorithms that guarantee completeness and/or optimality, and thus more easily deployable in on-device or edge environments with limited resources.

As in Experiment 1, we use $N=16$ for the best-of-N-chains method. For beam search, we use a beam width of 4, as this results in the same number of nodes per level in the search tree. (For beam width 4, we generate 4 candidates for each node in a level, then prune the level to 4 nodes and repeat, resulting in $4 * 4 = 16$ nodes generated per level.) To fit within our compute constraints, we restricted this experiment to $n=500$ samples from the GSM8K test split.

3.3 Experiment 3: Compute Efficiency

Solution evaluation method(s): PRM, Self consistency

Variable(s): Number of candidate solutions per solution step

Measured: Solution accuracy

This experiment investigates the effectiveness of PRM vs. self consistency across a range of compute budgets, in terms of the number of candidate solution chains sampled from the generator. The goal is to examine whether there is a compute budget for which majority voting in self-consistency can approximate the effectiveness of PRM guidance in selecting the best candidate

solution. Due to the large amount of solution generation, we limit this experiment to a subset of 250 problems from GSM8K instead of the full dataset.

4 Results

4.1 Experiment results

4.1.1 Experiment 1: Baseline comparisons

Evaluation Method	Accuracy
Greedy (1 inference)	0.199
Self-Consistency ($N = 16$)	0.333
PRM (best of $N = 16$)	0.404

Table 1: Comparison of final solution accuracy between PRM and baselines

Over the entire test set of 1319 problems, best-of-N PRM-guided inference outperformed naive unguided inference (i.e. inference without step verification/supervision), seeing a roughly 21% increase in accuracy.

4.1.2 Experiment 2: Search method comparisons

Over 500 problems, our default best-of-N approach ($N = 16$) outperformed beam search (beam size $K = 4$) for solution step selection. Specifically, beam search achieved 53.80% accuracy compared to best-of-N’s 58.20%.

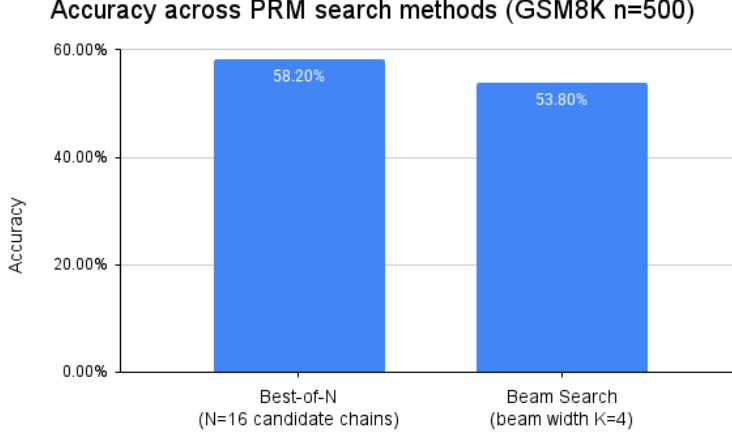


Figure 2: Case Study – Failure from frequent but incorrect answers

4.1.3 Experiment 3: Compute Efficiency

Over 250 problems, best-of-N PRM-guided inference performed better than self-consistency inference given the same compute budget N , i.e. the number of generated candidates per solution step. The accuracy gains are most pronounced for $N = 2, 4$, or 8 , with a maximum 27% increase in

accuracy over self-consistency for $N = 4$. The two methods hit similar rates of diminishing returns past $N = 8$, but PRM still maintains at least 0.05 additional accuracy for the remaining range of compute budgets we tested.

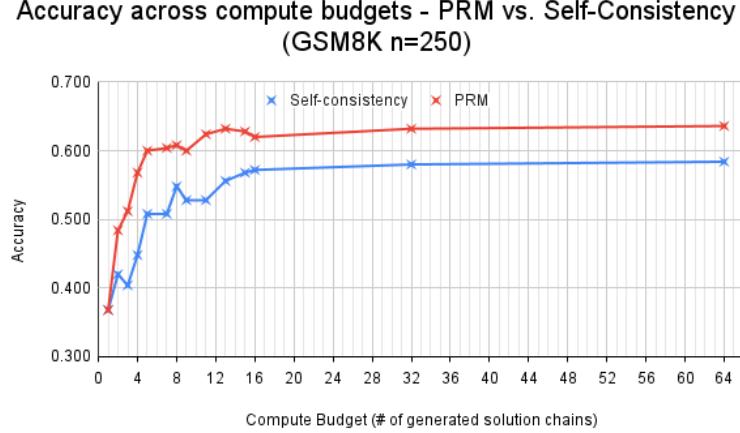


Figure 3: Case Study – Failure from frequent but incorrect answers

5 Discussion

5.1 Experiment 1: Baseline comparisons

PRM-guided inference resulted in 0.404 accuracy, a $\sim 21\%$ improvement compared to self-consistency inference (0.333 accuracy) with the same $N = 16$ candidate chains. (For reference, a single greedy inference attempt had an accuracy of 0.198, though we draw no further conclusions from this as the single inference instance is not directly comparable.) While the accuracy of 0.404 is far from the state of the art performance provided by large scale models, this is still a significant improvement over baseline for a tiny 0.6B parameter generator. The step-wise evaluation provided by the larger, more capable PRM is able to guide the smaller generator towards the most plausible solution, in a way that self-consistency, which relies solely on the smaller generator’s capabilities, is unable to achieve. Examining sample problems that were solved by PRM but not by self-consistency reveals two common failure modes that PRM addresses:

Frequent but incorrect answers Self-consistency models rely on majority voting. Due to their limited reasoning capabilities, smaller models such as the 0.6B parameter generator may generate incorrect solution steps more frequently than correct ones, meaning majority voting will discard the correct answer and select the more frequent, but incorrect solution chain instead. With PRM-guided best-of-N generation, the most plausible reasoning chain is always selected, regardless of how frequently similar chains are generated.

Tie breaking Similarly, when faced with a tie in votes, self-consistency models adopt arbitrary tie-breaking mechanisms and may not always choose the most plausible option. In this case, PRM scores act as an informed tiebreaker. If two chains are equally frequent, the PRM score guarantees selection of the most plausible chain.

"Marcell and Beatrice are having a contest to see who can eat the most fruit roll-ups, so they unroll as many as they can find. Unfortunately, someone makes a mistake and Beatrice's was two roll-ups wide and 24 rolls up long while Marcell's was 3 roll-ups wide and 14 roll-ups long. If they both ate their entire amount, how many did they eat on average?"

Correct answer: $(2 \times 24 + 3 \times 14) / 2 = 45$

Generator answers:

'30' x 3 (PRM scores: [0.393, 0.393, 0.559])

'45' x 2 (PRM scores: [0.859, 0.812])

'90' x 2 (PRM scores: [0.301, 0.017, 0.242])

Majority vote: '30'

PRM choice, Best of N=8: '45'

Figure 4: Case Study – Failure from frequent but incorrect answers

5.2 Experiment 2: Search method comparisons

We also compared different search methods for PRM-guided inference. We found that best-of-N sampling achieved slightly higher accuracy than beam search, but not by much (0.05 accuracy points). For our experiments, we compared N=16 and beam width K=4 to match the number of internal nodes computed in the search trees. However, Best-of-N does not perform per-step search. Given a larger beam width, beam search could explore a larger solution space while still maintaining efficiency from pruning; thus, it is likely that beam search with a larger K (e.g. K=16) could achieve results on par, if not better than best-of-N sampling, at not much higher cost.

5.3 Experiment 3: Compute Efficiency

Finally, given the same number of candidate solutions generated for each solution step, PRM-guided inference performed better than self-consistency inference, with a maximum 27% additional accuracy for $N = 4$. Even after hitting diminishing returns, PRM still maintains roughly 9% additional accuracy over self-consistency. The external evaluation provided by the PRM supplements the limited reasoning capabilities of the small generator model, improving performance on complex, multi-step reasoning tasks without increasing the compute budget or complexity of the generator model. This suggests that a weaker generator can be standalone for general purpose tasks, while a PRM can be reserved for supplementing more complex or multi-step reasoning tasks.

6 Limitations and Future Work

A primary limitation of our current approach is the reliance on the larger Qwen3-8B verifier during inference. While running a larger model for scoring introduces computational overhead, this design remains practical for real-world deployment through asynchronous verification. In such a setup, the lightweight generator provides low-latency initial responses, while the verifier operates in the background to validate or re-rank solutions without blocking the user interaction. This allows the system to maintain the perceived responsiveness of a 0.6B model while leveraging the reasoning depth of an 8B model.

However to achieve truly efficient on-device generation, future work must focus on distilling the step-by-step reasoning capabilities of the strong verifier back into the weak generator. This could be achieved through Reinforcement Learning (RL) techniques, such as Group Relative Policy Optimization (GRPO) [9], which would allow the generator to internalize the verifier’s feedback signal. Successfully aligning the generator’s policy would reduce or eliminate the reliance on an external verifier at inference time.

Additionally, our current evaluation is limited to the GSM8K dataset. It remains an open question whether a weak generator has the capacity to explore the solution space effectively for more abstract or complex reasoning tasks. Extending this framework to more challenging benchmarks, such as the MATH dataset [2], would be critical to rigorously test the limits of the “weak generator, strong verifier” paradigm.

7 Conclusion

This project sought to investigate how Process Reward Models (PRMs) can be used to enhance mathematical reasoning capabilities of tiny generator models suitable for deployment on edge devices. We demonstrate that with a PRM-informed Best-of-N strategy, we can significantly outperform the self-consistency baseline, achieving a $\sim 7\%$ increase in accuracy from 33.3% to 40.4%. We also show that best-of-N sampling achieves higher accuracy than a beam search with the same rough compute cost, and that PRM yields higher accuracy than self-consistency across a range of compute budgets (candidate solution chains).

Our findings show that the step-wise verification provided by the PRM can mitigate common failures in smaller models, such as issues with inaccurate but high-frequency predictions and tiebreaking. This means that a weaker generator can be used standalone for general purpose tasks, and supplemented by supervision from a stronger PRM for more complex mathematical (or general multi-step reasoning) tasks.

Ultimately, this study suggests that intermittent oversight from a strong verifier can efficiently extract extra reasoning capabilities from a weak model. In turn, this shows that investing in step-level annotated datasets is justified, as its usage in an external verifier yields improved accuracy without any additional modification of the on-device model.

8 Code and Data Availability

To facilitate reproducibility, we have made our complete source code, including detailed documentation and the notebooks used to replicate the experiments, publicly available on GitHub at:

<https://github.com/devangb3/Process-Reward-Models>

Additionally, the model weights used in this study are hosted on Hugging Face:

- **Fine-tuned PRM Verifier:** devangb4/prm-qwen3-8b-bf16-6k
- **Generator Model (Qwen3-0.6B):** Qwen/Qwen3-0.6B

Details regarding the datasets used for training and evaluation can be found within the linked repository.

9 Author Contributions

The authors contributed to this work as follows:

- **Devang Borkar:** Developed Process Reward Model verifier and implemented LoRA fine-tuning on PRM800K. Designed and implemented Experiment 1.
- **Kwabena Manu:** Designed inference algorithms and built the integration pipeline between PRM and generator. Designed and implemented Experiment 2.
- **Linus Lam:** Designed and implemented Experiment 3. Performed the statistical analysis, case studies, and visualization of the results.

All authors contributed equally to the drafting, reviewing, and editing of the final manuscript.

References

- [1] 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. *arXiv preprint arXiv:2110.14168*, 2021.
- [2] 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. *NeurIPS*, 2021.
- [3] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [4] 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. *arXiv preprint arXiv:2305.20050*, 2023.
- [5] 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, 2022.
- [6] 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, 2023.
- [7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS ’22, Red Hook, NY, USA, 2022. Curran Associates Inc.
- [8] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen3 technical report, 2025.
- [9] Qihao Zhu Runxin Xu Junxiao Song Mingchuan Zhang Y.K. Li Y. Wu Daya Guo Zhihong Shao, Peiyi Wang. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.